# Nordic Blockchain

**Semi-Centralized Distributed Ledger Technology for banks**

**Luca Francioni (240068)**

**Supervisor: Michael Viuff**

**VIA University College - Horsens**

**39,265 Characters**

**ICT Engineering**

**7th Semester**

## Table of content

## Abstract

Nordic Blockchain is a software service for banks, its purpose is to replace previous mechanisms and systems used in banks for transactions with a modern blockchain based one, reducing costs of operations and improving inter-banks data synchronization and trust; it offers a backend oriented only plan, as the project aims to become a starting point for future projects based on the same logic of this.
The following report describes the analysis, design and implementation of such service but properly distinguishing the intended design and the resulting prototype. The project also raises technical and security aspects that can be subject to academic scrutiny and further development.
The appendix section contains the full code of the service; documentation towards the modular frameworks and components being used are described in the document.
The service is structured with the intent of being easily understandable, replicable and extendable.

# 1 Introduction

The banks' services have been alive and used for millennia, being the core structure of the worldwide economy development and actively participating in assets exchange, monetary value fluctuation, investments and everyday money exchange for services.

The transfer of assets of a bank's customer to another bank's customer is a process that involves several systems – either human or technology based - that must be properly configured and synchronized, offering as well adequate security and control for the bank's administration, transparency and certainty for both the banks involved that the transactions are valid, secure and effectively requested by the originating customer: this may lead to slow downs and the overall system must always be available, scalable and stable.

The current systems involve years of software and hardware development and research, making a user operation fee based his/her desire of guaranteed speed of processing for the operation to be completed and the complexity of the transaction request path (e.g. international transactions), yet faster than physical writing and human elaboration, but still exposing the mechanism to exploits by malicious individuals (e.g. frauds, fake signatures, system penetrations, phishing, etc.).

One of the most notorious systems being used is SWIFT, a global cooperative owned software used for secure financial messaging services; the software is from 1974 and subject to millions of transaction requests to process worldwide, considered the current best due to high security standards, better bandwidth usage and high monitoring capabilities.

The blockchain technology is a structure design oriented to continuous security integrity, transparency and independence of a central system for administration; such technology is specifically designed to be reliable at any moment and ensures all involved parties that the data stored is permanent and not tamperable.

Moreover, the blockchain is a very simple and light structure that can be easily customized by the needs, featuring a huge global open source community of volunteers constantly proposing improvements and extensions to the security, storage and transmission components; offering a significant margin of improvement over the time.

Blockchains may change the way the whole underlying structure of a bank transaction management system work, unlocking unprecedented levels of scalability, speed of the operation and its costs, including security and transparency between multiple banks with absolute certainty of validity of the data by replacing the traditional systems with a modern blockchain design. Aiming to be the most independent and collaborative middle-ware system for transactions of assets and multiple possible operations that need to be synchronized between multiple banks in real time with least effort possible and maximum speed of transmission available; putting the storage control system software in each bank internal system and allowing banks to run undefined number of elaboration software components to process the data with fraud and intrusion detection algorithms, including already adopted ones with a plug-and-play design.

If properly adopted, the design, can be integrated or be the replacement, to the current software systems without the bank's customers notice as mimicking the same current functionalities while radically changing the mechanism behind it including, and not only to, the quality of life and service.

Such newly adopted design through, will have to reinvent parts of the original concept of blockchains such as transaction processing and the decentralization nature.

The SWIFT technology may feature modern and state of the art technologies for transaction elaborations and security monitoring, such technology is, however, not using publicly known structure but is instead using RESTful APIs architecture, making the system especially centralized and closed and/or slowing the process of access another bank's transactions logs.

The usage of a blockchain, a chain of "blocks", containing transactions and being synchronized in real time by all the blockchain's network nodes (Figure 1), completely removes slowdowns of accessing certain essential information from the monitoring systems and significantly speeds up the process of a transaction request, as well as offering flexibility for temporary offline nodes, that only have to download latest version of the blockchain to get back to operative status.

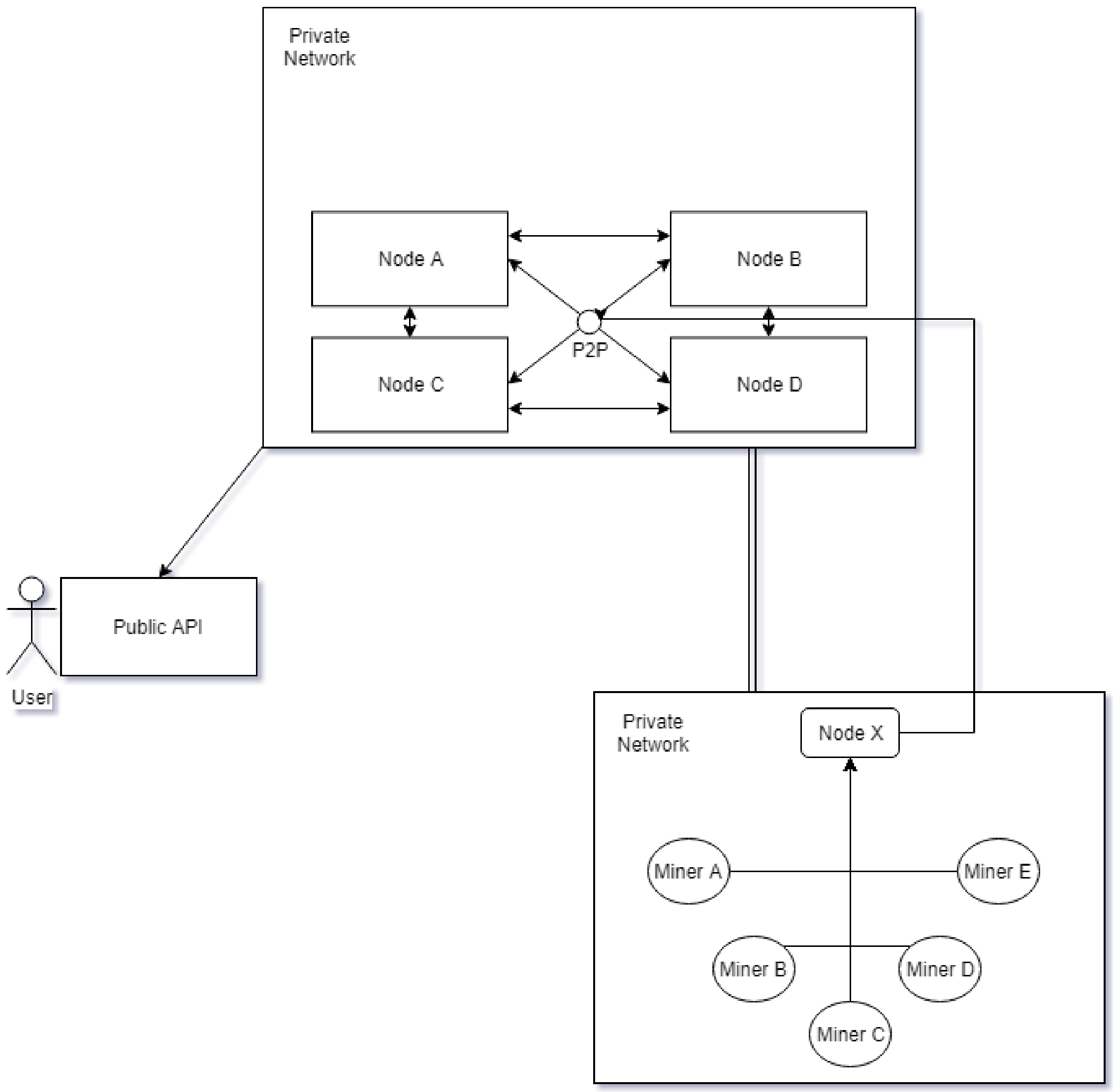


*Figure 1 - The overview of a blockchain network*

In a blockchain, the key role of the bank becomes the node, the node is a copy of the blockchain that continuously synchronizes its content over the time, becomes the single bank's internal transaction management system to which its customers transactions pass through and queued in the miner's pending operations. The node is also the internal monitor system, as it checks the validity of the transaction requests' origin before being inserted into the queue.

The nodes are not able to communicate to each other directly and transactions aren't immediately registered into the blocks of the blockchain.
This introduces the known figure of the miner: its original schedule – using Bitcoin as comparison reference – is to discover new block trough specific cryptographic operations, where new transactions are going to be stored between the nodes; and the verification process that the transaction is valid and trustworthy considering the origin address and the destination address.

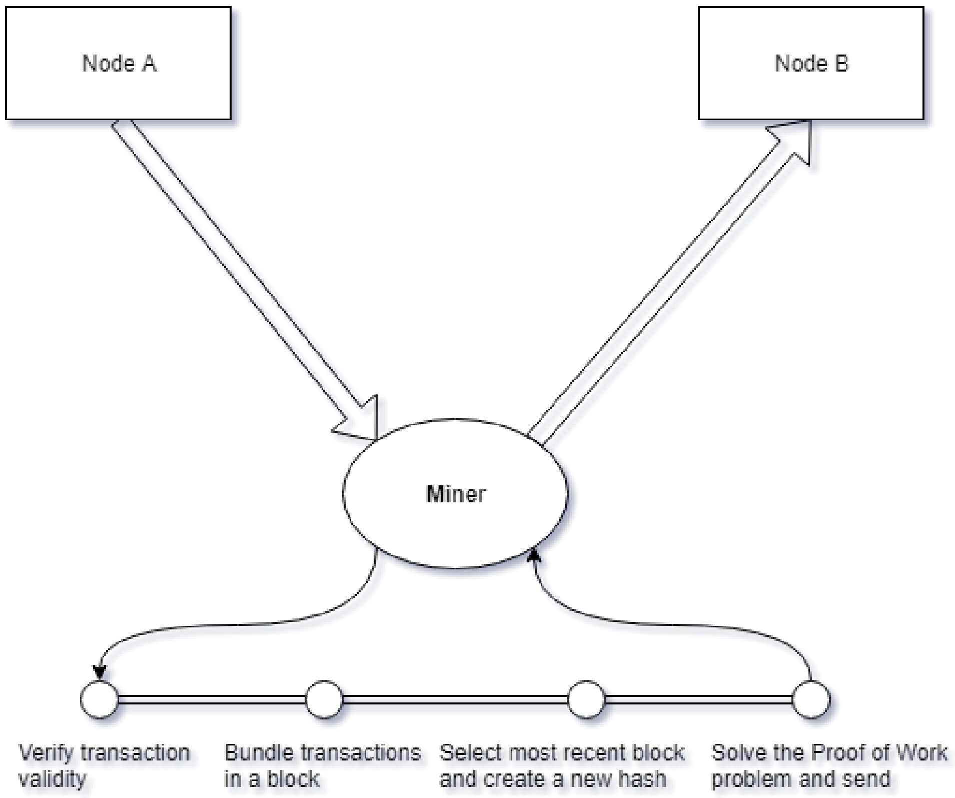

Node A
Node B
Miner
Verify transaction validity
Bundle transactions in a block
Select most recent block and create a new hash
Solve the Proof of Work problem and send

*Figure 2 - The miner's process of verification of a transaction*

In Bitcoin’s blockchain, the miner’s role (Figure 2) is to verify the validity of the transaction, bundle the elaborated transactions in a block, calculate the new block hash and insertion and solve the “Proof of Work” problem to demonstrate the miner’s reliability and eventually let the newly inserted block propagate trough the blockchain’s network. The role of the miner is usually following the same general logic in all the blockchains, but different distribution usually adopts different processes for the same results.

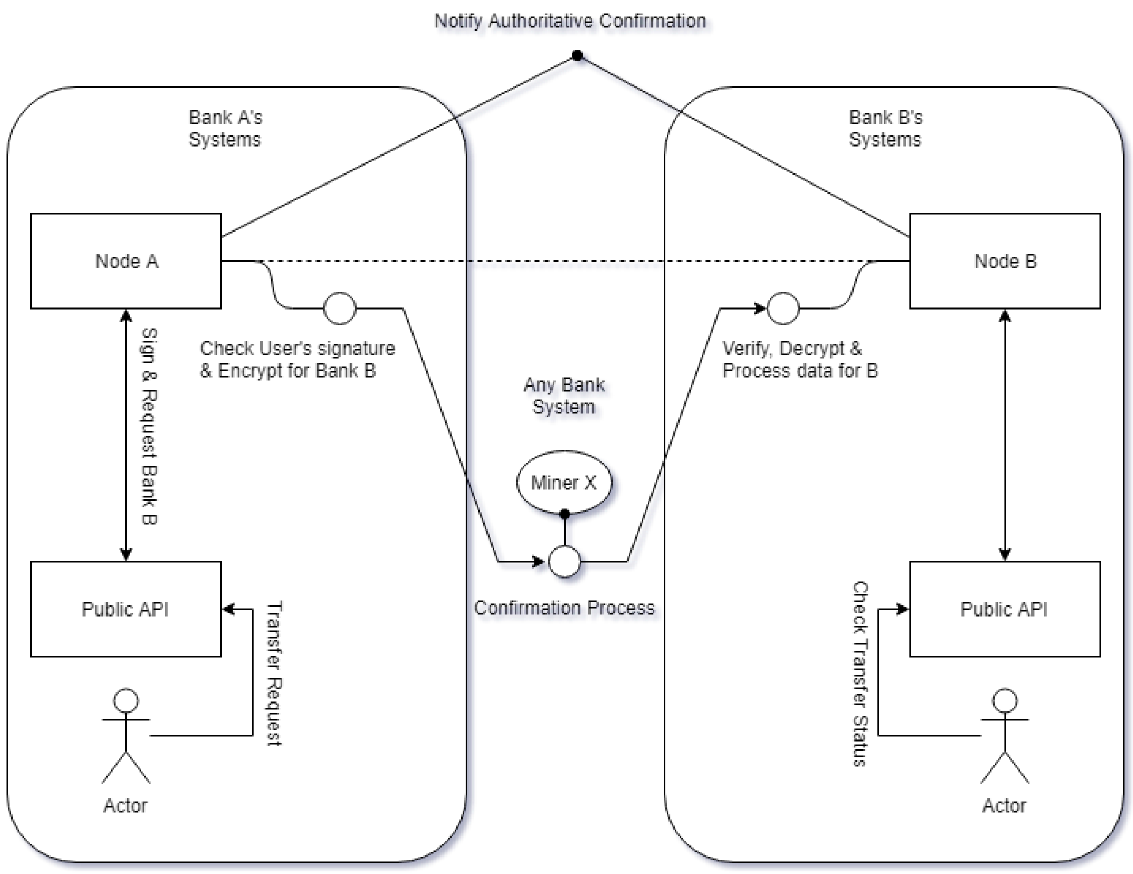


*Figure 3 - Miner's macro-role in the blockchain network*

When a miner ends its verification process, a verdict is sent to the blockchain network by increasing the transaction’s synchronized confirmation counter by 1.

Once the confirmation counter of a transaction reaches a fair amount, the transaction is to be considered trustworthy, hence, confirmed and concluded (Figure 3).

The process above described may appear long, but given the correct setup of the nodes and resources allocated to the software and enough miner being run by the collaborative network, the transaction time can be drastically cut to minutes (internationally) and the costs of the transaction fees is reduced as well, due to a significant cut of bureaucracy processes that can now be slightly adapted to a natively semi-transparent system; therefore the more miner are deployed in the internal network the faster the transactions are processed, unlocking a significant scalability solution by simply deploying more instances on running systems.

As of the monitoring, the miner's process can be easily customized by adding custom steps, these can be also the current existing monitoring platforms, allowing perfect modularity and extension of the whole Nordic Blockchain platform with little to no risk of unintentional sabotage/misconfiguration and an easier migration from previous platforms to the proposed one.

# 2 Analysis

## 2.1 Requirements

The following are the requirements adopted for this project.
The involved users in the various requirements are mainly of three types:

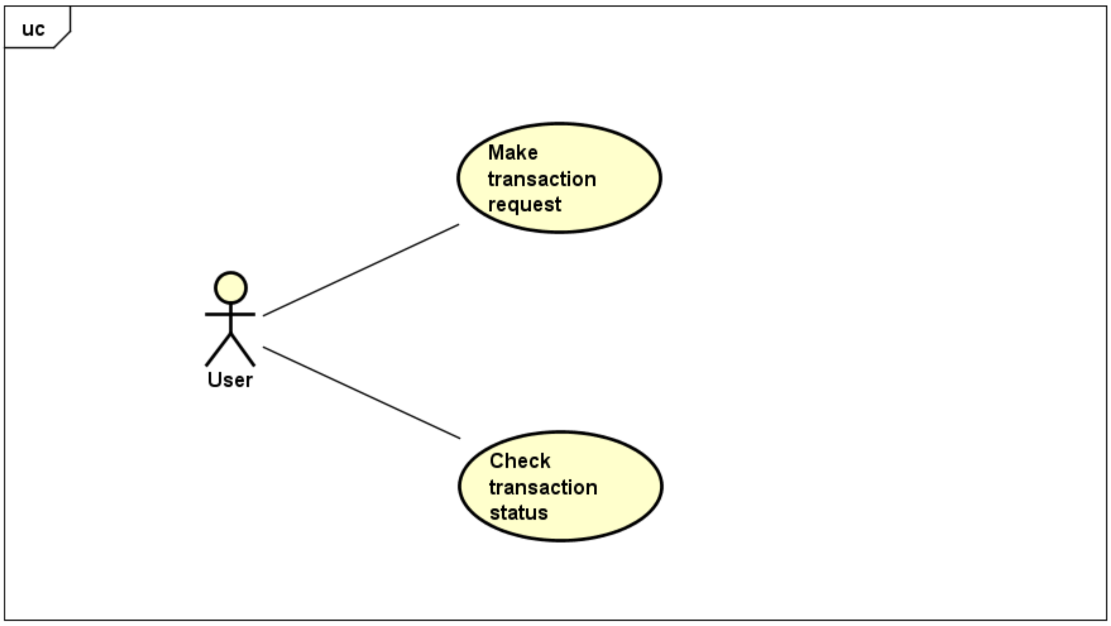
- **User** – A bank customer, average user that has no knowledge of what is happening behind the scene.

- **Bank Operator** – A bank employee dedicated to the project, depending on its goal.

- **Miner** – An automated system for transactions confirmation, that ensures security and validity.

## 2.2 Functional Requirements

### a) User must be able to interact with transactions.



- **Make transaction request**

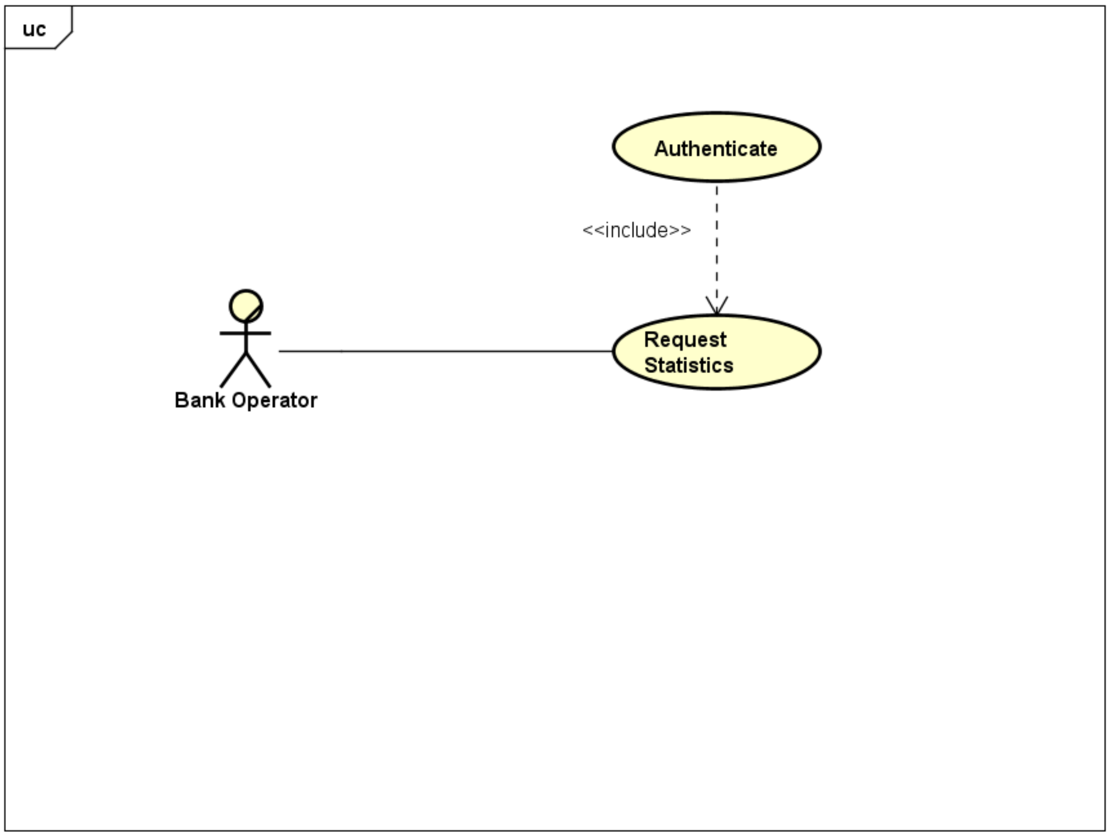
| | |
|---|---|
| **Summary** | The user wishes to send money to a friend, such user must therefore create a transaction request trough the his/her’s bank application. |
| **Precondition** | The user has the receiver address and knows the asset value (monetary amount) of the transaction. |
| **Postcondition** | Success: the transaction request creation process gives back a transaction identifier.<br>Failure: no transaction identifier is given. |

| | |
|---|---|
| **Base Sequence** | 1. User specifies receiver.<br>2. User specifies the author (itself).<br>3. User specifies the asset value.<br>4. Create transaction request and approve.<br>5. The system acquires the transaction request and puts it into processing queue. |

- **Check transaction status**

| | |
|---|---|
| **Summary** | The user wishes to check the previous transaction status, if it has been elaborated or refused; trough the bank's application. |
| **Precondition** | The user has the transaction identifier given by the transaction request creation and confirmation. |
| **Postcondition** | Success: the user can see the transaction's status information.<br>Failure: no information found. |
| **Base Sequence** | 1. User specifies transaction identifier.<br>2. User sends request.<br>3. User obtains result, if successful. |

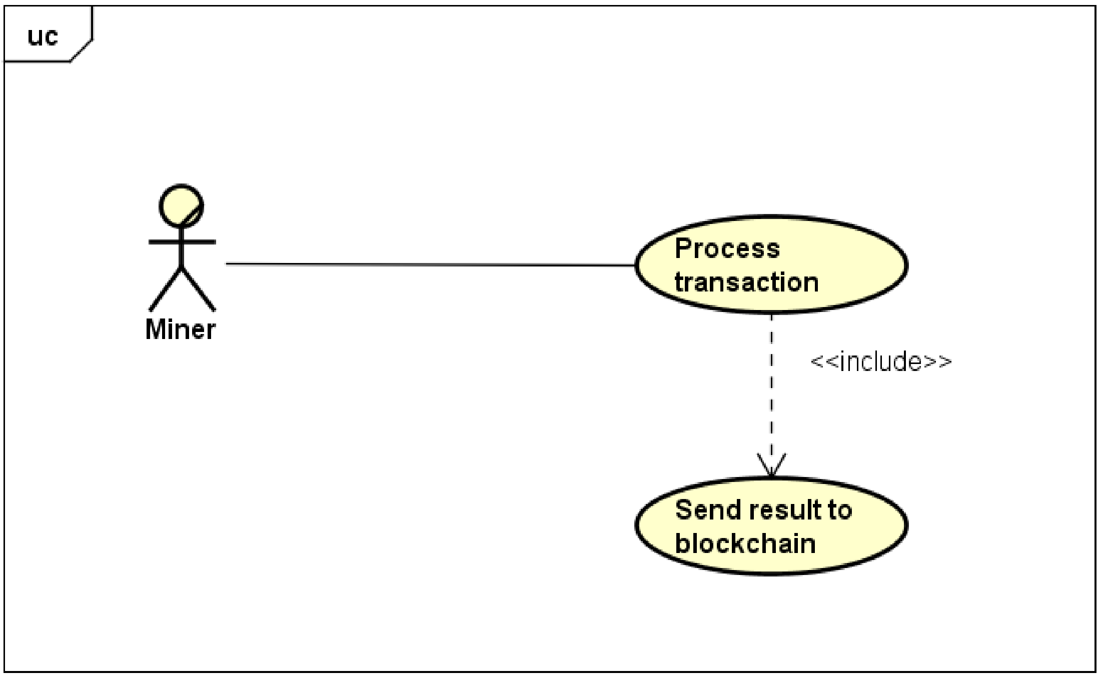
**Blockchain statistics must be available to operator**



| | |
|---|---|
| **Summary** | A bank operator wishes to check the blockchain's latest information for monitoring. |
| **Precondition** | The operator is an authorized individual with a valid authentication key and a network connection to the node. |
| **Postcondition** | Success: the operator obtains the statistics.<br>Failure: no response from the node. |
| **Base Sequence** | 1. Operator creates a statistics request.<br>2. The system acquires a copy of the statistics to send. |

| | |
|---|---|
| | 3. Operator obtains the information requested. |

### b) Transaction confirm



- **Process transaction**

| | |
|---|---|
| **Summary** | The miner is available to elaborate a transaction for an eventual confirmation or rejection. |
| **Precondition** | The miner has a network connection to the node. |
| **Postcondition** | Success: the miner obtains a pending operation to elaborate.<br>Failure: no response from the node. |
| **Base Sequence** | 1. Miner creates a pending operation request.<br>2. System checks if there are any pending operations in queue. |

| | |
|---|---|
| | 3. Miner receives a pending operation to elaborate. |

- **Send result to blockchain**

| | |
|---|---|
| **Summary** | The miner's processing result must be sent to the blockchain after the various checks. |
| **Precondition** | The miner already obtains a transaction identifier and already performed its checks and has a connection to the node. |
| **Postcondition** | Success: the operator obtains the statistics.<br>Failure: no response from the node. |
| **Base Sequence** | 1. The miner decides to cast a vote if operation is valid.<br>2. Send to the node the request of casting a vote by using such identifier.<br>3. The system acquires the vote and registers it. |

## 2.3 Non-Functional Requirements

- The transaction request transmission should happen within 2 seconds after compilation and confirmation of the operation.
- The resulting data from transaction status query must be transmitted to the requesting user within 10 seconds from the request.
- The response of the statistics query must happen within 10 seconds.

#### a) Performance

The Quality of Service offered by the software strictly depends on the performance of the system the bank decides to install.

a. Network – It is heavily encouraged to monitor bandwidth usage as a blockchain may have lots of network operations.
b. Virtual space – The fork of a blockchain may become bigger and bigger by the time, virtual space is the main resource that should be scalable.

**b) Operating constraints**

It is required by the whole bank systems to be connected to a common network for communication to work properly preferably using VPNs, ensuring also additional security and layer of isolation.

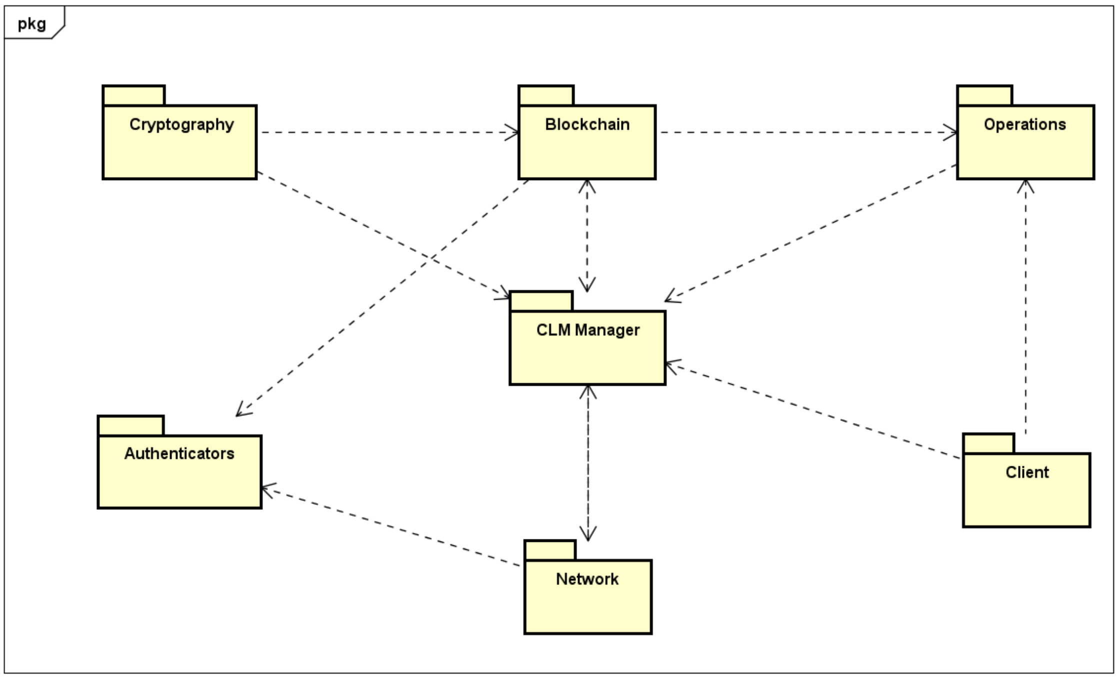
**c) Security**

The system must use known and approved cryptography mechanisms to guarantee security of the blockchain, the current asymmetric cryptography algorithm is RSA, can be upgraded to ECDH for better security measure. It is also necessary for production usage a layer of security for network communication protocol.

# 3 Design

The following section describes the main components of the software, highlighting relevant information for comprehension of the subject and the mechanism of functioning.

### 3.1.1 Simplified Overall Version

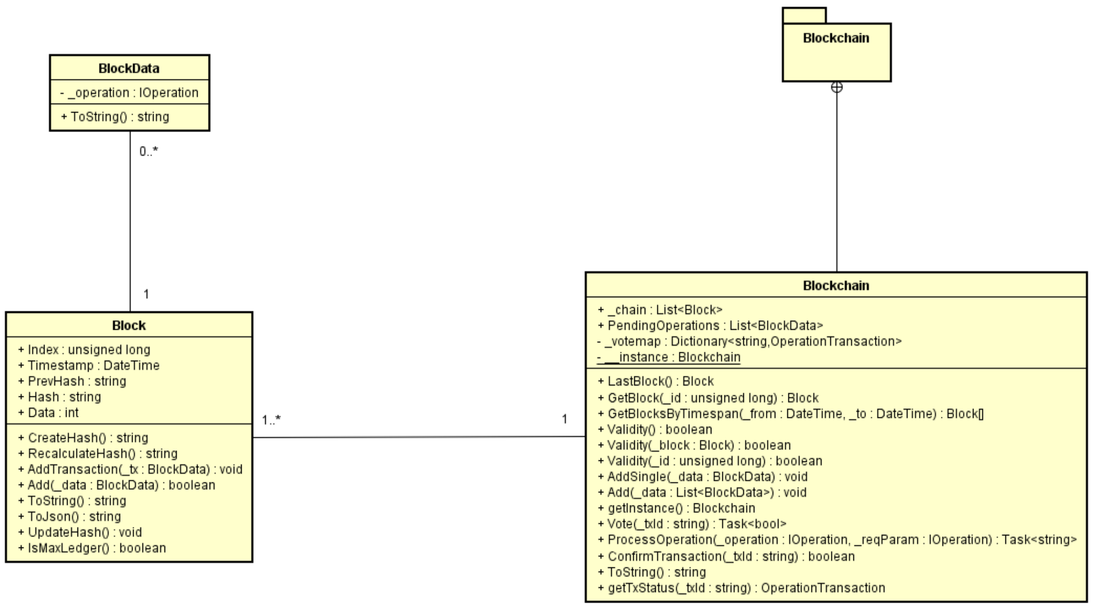


The whole Nordic Blockchain software is composed by its essential classes to which almost all depends on each other.

### 3.1.2 Full Overall Version

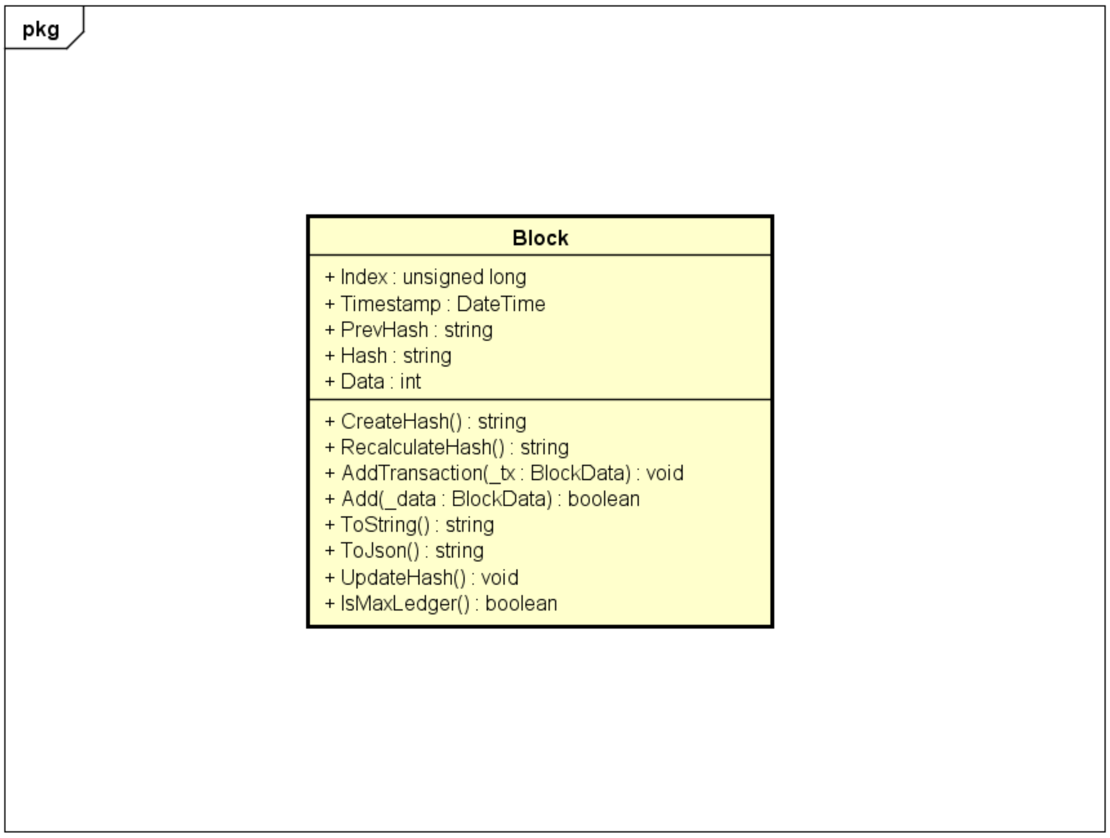
Refer to Appendices (Overall).

## 3.2 The blockchain


BlockData
- _operation : IOperation
+ ToString() : string
0..*
1
Block
+ Index : unsigned long
+ Timestamp : DateTime
+ PrevHash : string
+ Hash : string
+ Data : int
+ CreateHash() : string
+ RecalculateHash() : string
+ AddTransaction(_tx : BlockData) : void
+ Add(_data : BlockData) : boolean
+ ToString() : string
+ ToJson() : string
+ UpdateHash() : void
+ IsMaxLedger() : boolean
1..*
1
Blockchain
Blockchain
+ _chain : List<Block>
+ PendingOperations : List<BlockData>
- _votemap : Dictionary<string,OperationTransaction>
- __instance : Blockchain
+ LastBlock() : Block
+ GetBlock(_id : unsigned long) : Block
+ GetBlocksByTimespan(_from : DateTime, _to : DateTime) : Block[]
+ Validity() : boolean
+ Validity(_block : Block) : boolean
+ Validity(_id : unsigned long) : boolean
+ AddSingle(_data : BlockData) : void
+ Add(_data : List<BlockData>) : void
+ getInstance() : Blockchain
+ Vote(_txId : string) : Task<bool>
+ ProcessOperation(_operation : IOperation, _reqParam : IOperation) : Task<string>
+ ConfirmTransaction(_txId : string) : boolean
+ ToString() : string
+ getTxStatus(_txId : string) : OperationTransaction

The blockchain is the combination of stored blocks linked together and it’s the whole tree that composes the history of the confirmed operations inside the banks network. A blockchain may be reset (wiped clear) to a completely new one for virtual space usage optimization (and archiving backups), the version of a blockchain is called “Fork”.

### 3.2.1 Block



The block is the container of transactions being synchronized between multiple nodes. It is stored into the blockchain and locked for modifications once replaced by another block.

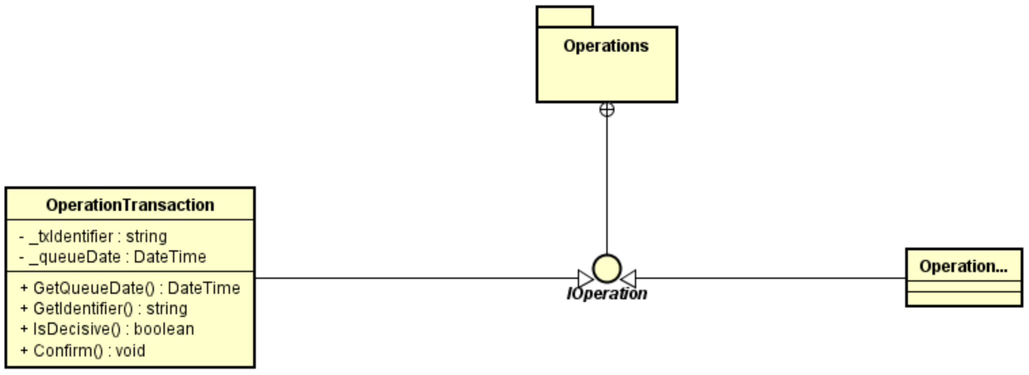
There exist two conceptual types of blocks:

- **Genesis block:** the first block of the blockchain, in the project it is the deposit of all of the trusted public keys for the bank's miners and other banks' nodes (further described below).
- **Normal block:** the normal block containing various block data (transactions).

### 3.2.2 BlockData

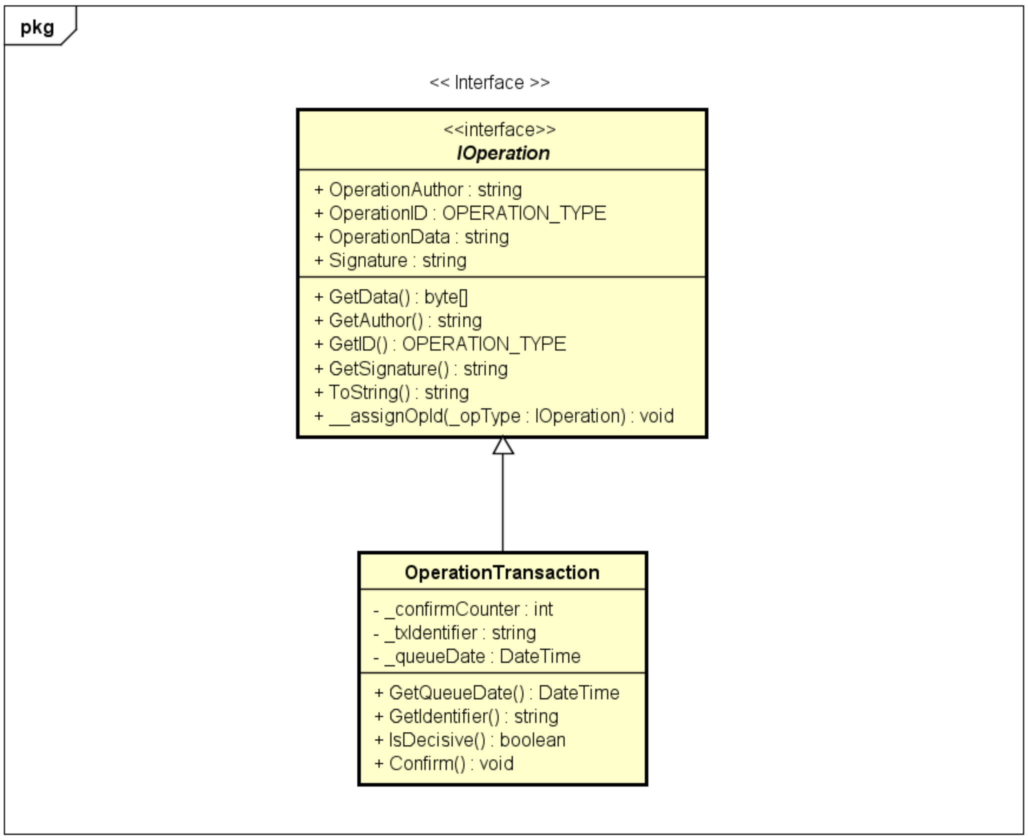
It is a container for IOperation stored in a Block.
It is required to contain only a reference to the interface instead of the specific implementing type of IOperation, so that all of the desired IOperation(s) are storable inside the BlockData.

## 3.3 The Operations



IOperation is an interface defining both network operations and Block's content.
It is handled both by the CLM Manager and stored in the BlockData.

### 3.3.1 OperationTransaction

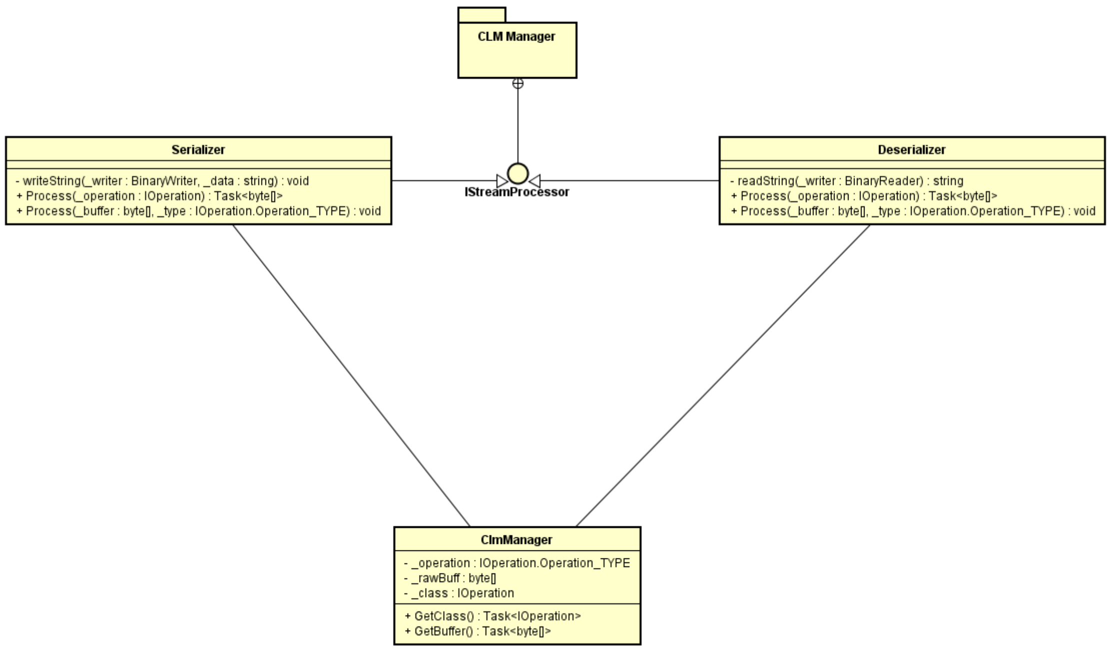


The transaction is the entity containing vital information for the asset transfer from a user to another, containing:

- **Sender Address** – The sender’s address (identifier).
- **Receiver Address** – The receiving user’s address (identifier).
- **Asset Value** – The value of the asset being transferred (e.g. money).

Such class also contains the transaction’s identifier and the counter of votes by the miners.
This class is also stored in the “_votemap” of the blockchain, for direct reference between TxID and the actual Tx (transaction) during the vote casting.

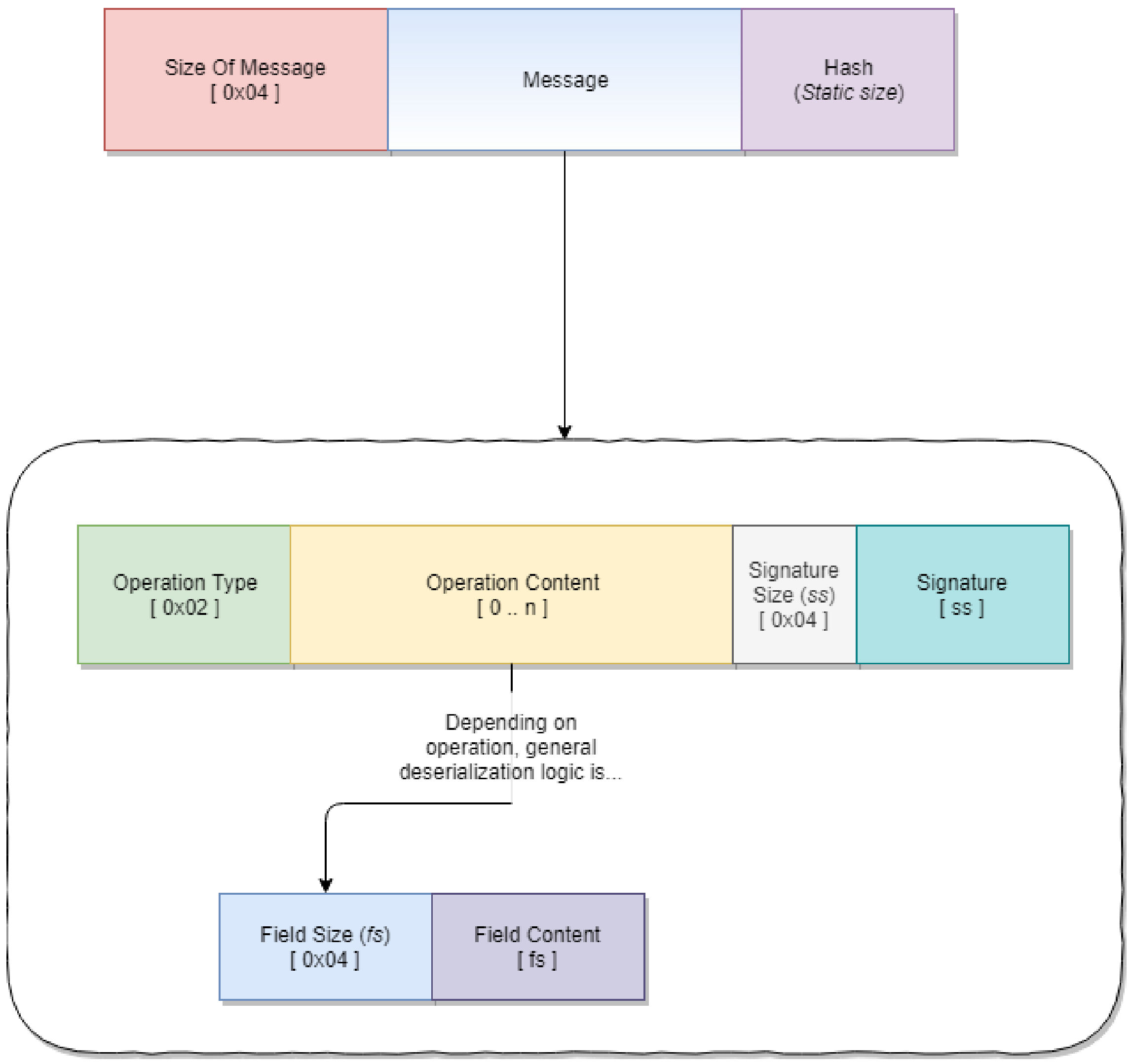
## 3.4 Crypto Layered Message (CLM Manager)


CLM Manager
Serializer
- writeString(_writer : BinaryWriter, _data : string) : void
+ Process(_operation : IOperation) : Task<byte[]>
+ Process(_buffer : byte[], _type : IOperation.Operation_TYPE) : void
IStreamProcessor
Deserializer
- readString(_writer : BinaryReader) : string
+ Process(_operation : IOperation) : Task<byte[]>
+ Process(_buffer : byte[], _type : IOperation.Operation_TYPE) : void
ClmManager
- _operation : IOperation.Operation_TYPE
- _rawBuff : byte[]
- _class : IOperation
+ GetClass() : Task<IOperation>
+ GetBuffer() : Task<byte[]>

The Crypto Layered Message Manager (CLM Manager) is the responsible class for deserialization and serialization of the CLM packets sent through the network.

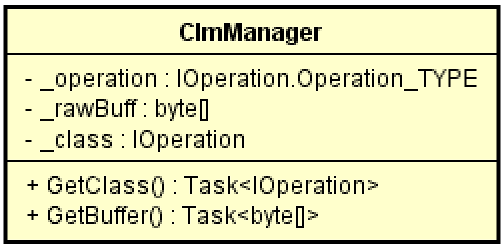
### 3.4.1 Crypto Layered Message

To ensure security and obscurity of vital information, the data being sent would be properly encrypted and signed through asymmetric encryption algorithms.
The encapsulation message is the container of the information being sent and the protocol to follow for correct requests and responses; structured in the following way:



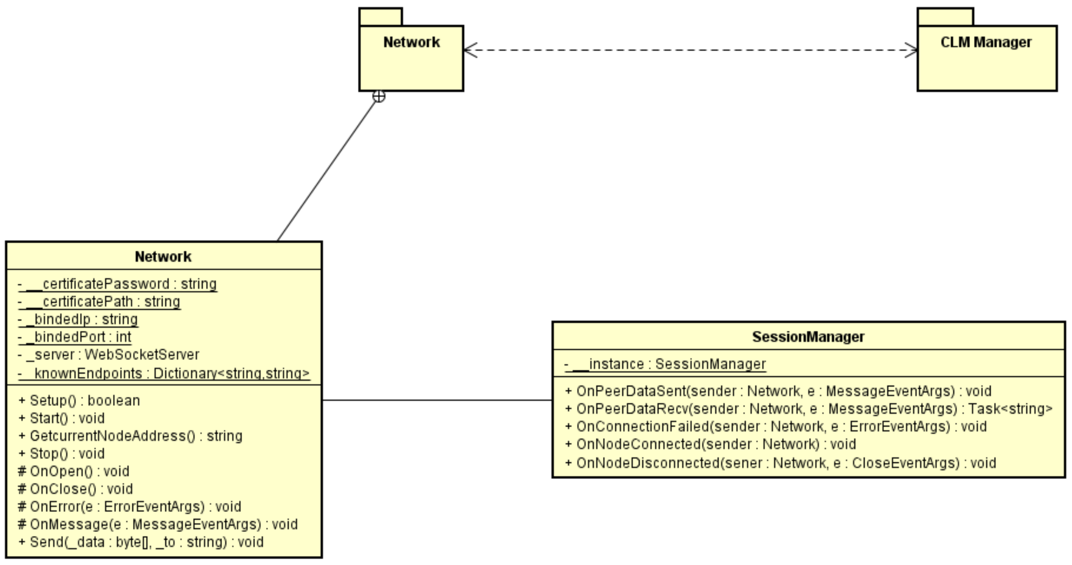
The CLM Manager employs a standard universal method for all the IOperation classes following this structure, since it obeys on a single interface.

### 3.4.2 CLM Manager


ClmManager
- _operation : IOperation.Operation_TYPE
- _rawBuff : byte[]
- _class : IOperation
+ GetClass() : Task<IOperation>
+ GetBuffer() : Task<byte[]>

The CLM Manager must take as input either a buffer or a class and transform it into the related opposite (from class to buffer, from buffer to class).

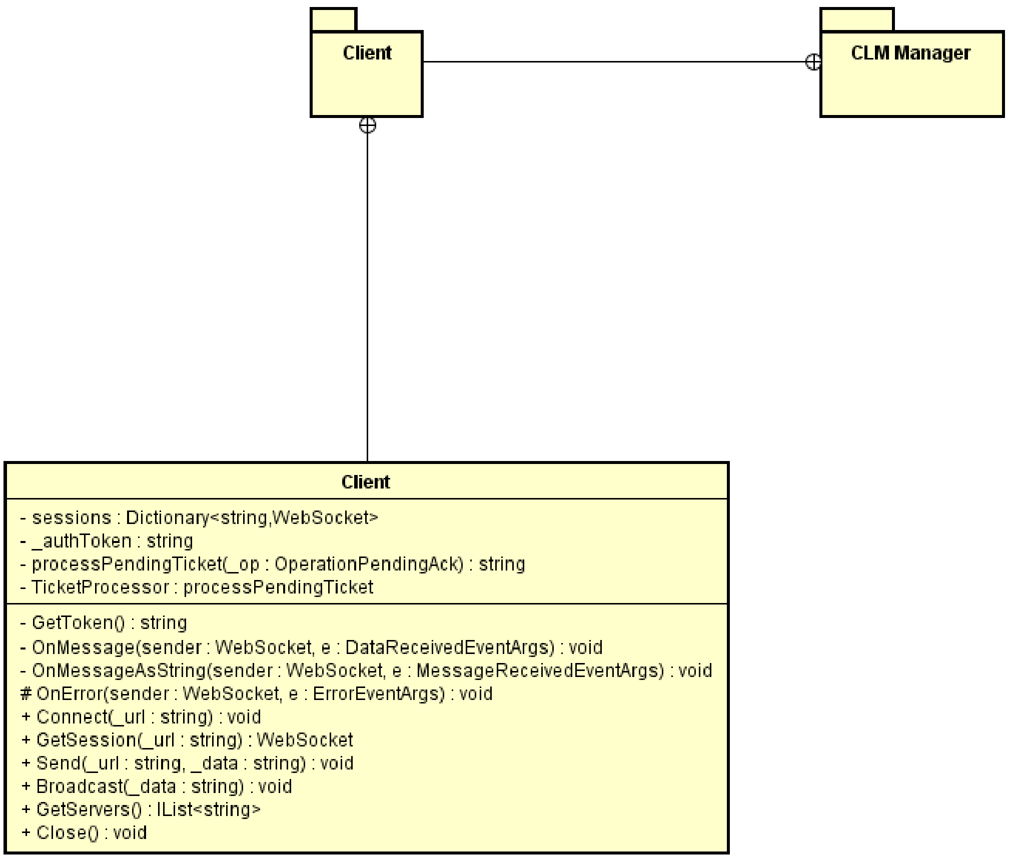
## 3.5 Network


Network
CLM Manager
Network
- __certificatePassword : string
- __certificatePath : string
- _bindedIp : string
- _bindedPort : int
- _server : WebSocketServer
- _knownEndpoints : Dictionary<string,string>
+ Setup() : boolean
+ Start() : void
+ GetcurrentNodeAddress() : string
+ Stop() : void
# OnOpen() : void
# OnClose() : void
# OnError(e : ErrorEventArgs) : void
# OnMessage(e : MessageEventArgs) : void
+ Send(_data : byte[], _to : string) : void
SessionManager
- __instance : SessionManager
+ OnPeerDataSent(sender : Network, e : MessageEventArgs) : void
+ OnPeerDataRecv(sender : Network, e : MessageEventArgs) : Task<string>
+ OnConnectionFailed(sender : Network, e : ErrorEventArgs) : void
+ OnNodeConnected(sender : Network) : void
+ OnNodeDisconnected(sener : Network, e : CloseEventArgs) : void

The "Network" class is responsible for setting up the server and handle incoming transmissions of data, it is the entry point for all the IOperation(s).

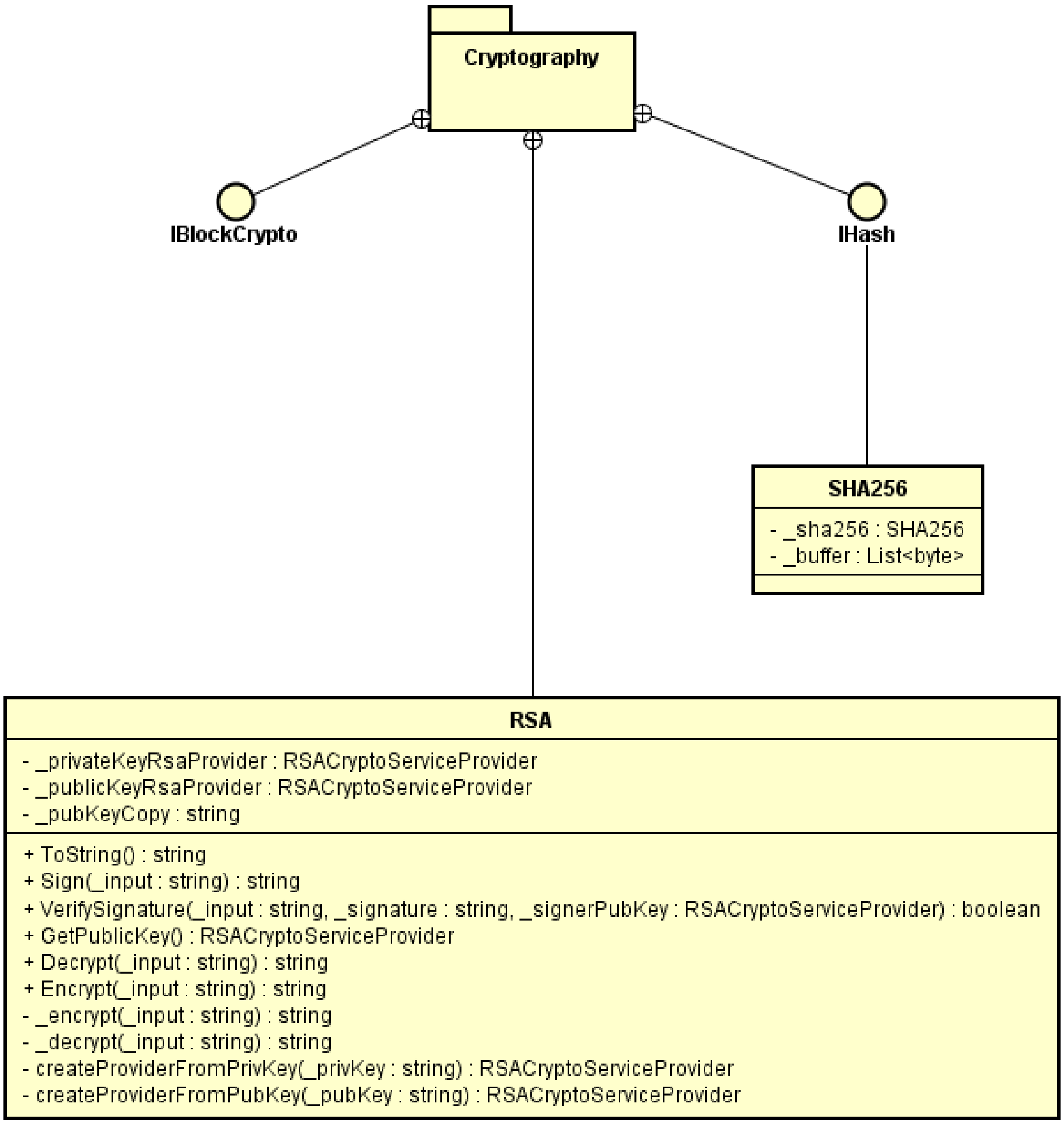
The server is a WebSocket with SSL on top, the transmitted data is encoded in base64 to avoid character encoding complications.

### 3.5.1 Client


Client
CLM Manager
Client
- sessions : Dictionary<string,WebSocket>
- _authToken : string
- processPendingTicket(_op : OperationPendingAck) : string
- TicketProcessor : processPendingTicket
- GetToken() : string
- OnMessage(sender : WebSocket, e : DataReceivedEventArgs) : void
- OnMessageAsString(sender : WebSocket, e : MessageReceivedEventArgs) : void
# OnError(sender : WebSocket, e : ErrorEventArgs) : void
+ Connect(_url : string) : void
+ GetSession(_url : string) : WebSocket
+ Send(_url : string, _data : string) : void
+ Broadcast(_data : string) : void
+ GetServers() : IList<string>
+ Close() : void

The “Client” class is responsible for the client connection to the blockchain’s servers, handle IOperation(s) and automatically handle response to these.

## 3.6 Cryptography


Cryptography
IBlockCrypto
IHash
SHA256
- _sha256 : SHA256
- _buffer : List<byte>
RSA
- _privateKeyRsaProvider : RSACryptoServiceProvider
- _publicKeyRsaProvider : RSACryptoServiceProvider
- _pubKeyCopy : string
+ ToString() : string
+ Sign(_input : string) : string
+ VerifySignature(_input : string, _signature : string, _signerPubKey : RSACryptoServiceProvider) : boolean
+ GetPublicKey() : RSACryptoServiceProvider
+ Decrypt(_input : string) : string
+ Encrypt(_input : string) : string
- _encrypt(_input : string) : string
- _decrypt(_input : string) : string
- createProviderFromPrivKey(_privKey : string) : RSACryptoServiceProvider
- createProviderFromPubKey(_pubKey : string) : RSACryptoServiceProvider

The cryptography involved in the project. For asymmetric cryptography, RSA is the choice for this project; it is required that the class can load both private and public keys, sign and verify data.
SHA256 is the hashing algorithm adopted in this project since its popularity and wide usage due to its still valid security.

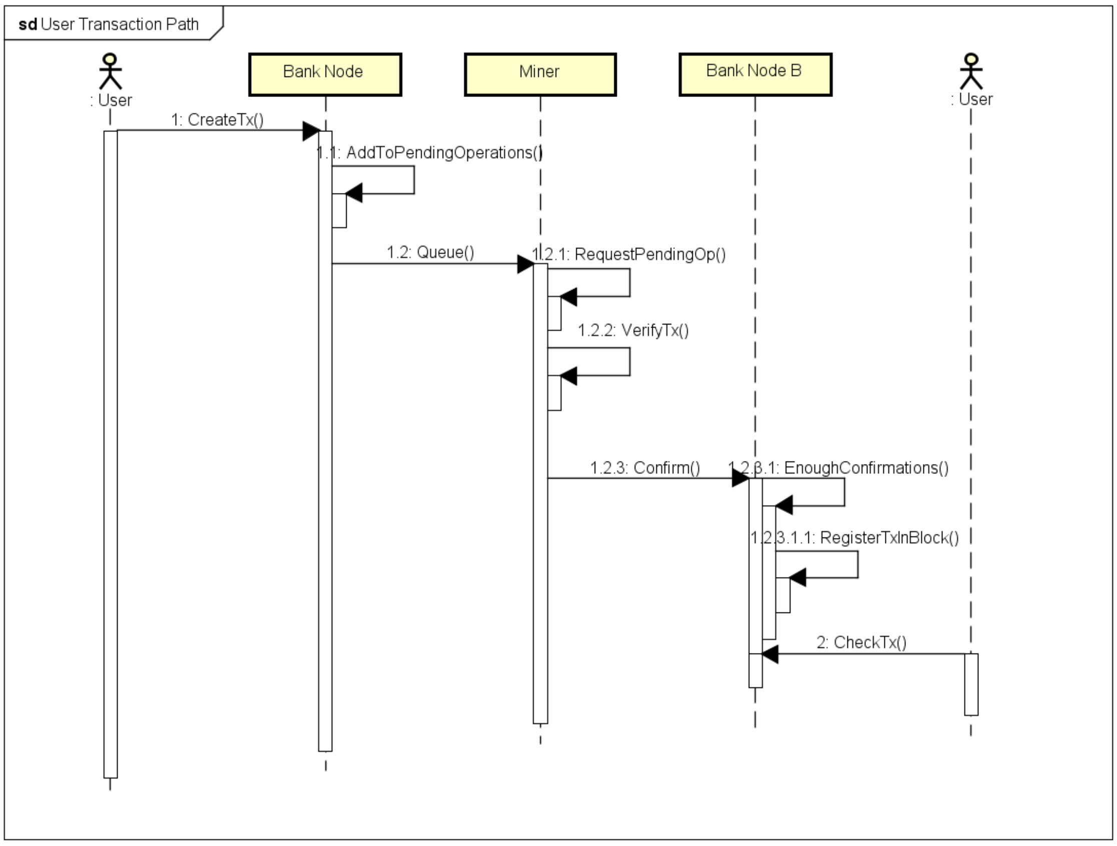
## 3.7 Other Entities

### 3.7.1 Node

The node is the central unit a bank uses for synchronization and storage of pending transactions originated by its own customers (users).
Every bank can have one or more nodes.
The node is structured by:

- Its own copy of the blockchain, that can be synchronized with other nodes.
- Its own customers pending transactions (awaiting miner's confirmation).
- Transactions confirmation counters.

### 3.7.2 Transaction Request Path



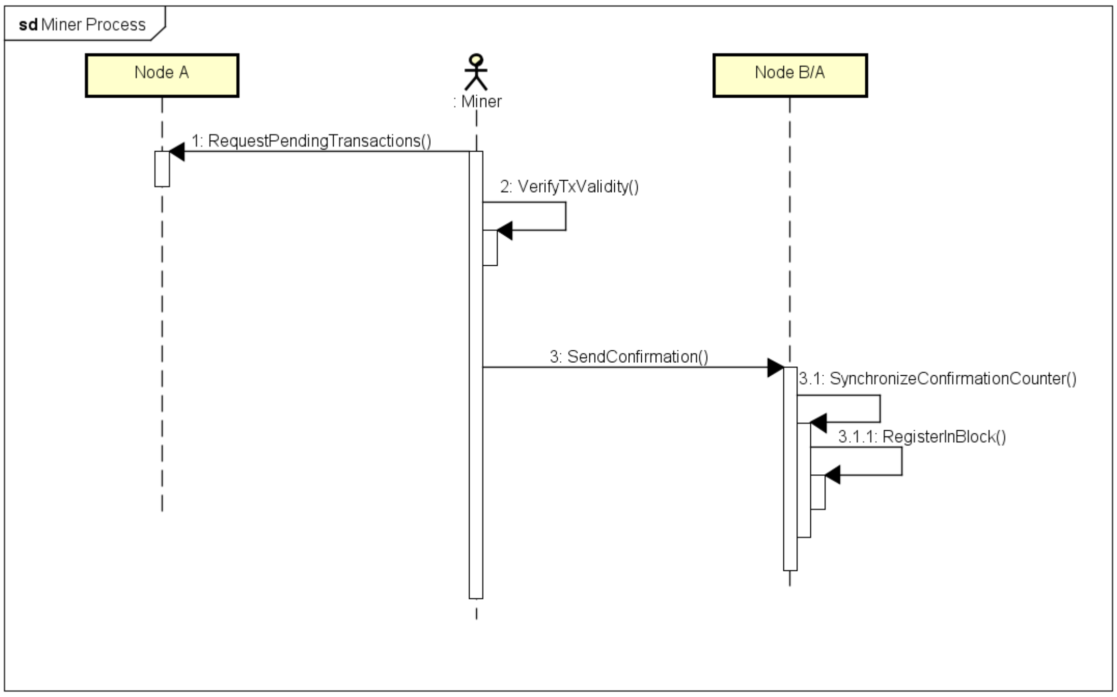
A transaction follows a long path of processing before being registered permanently for its scope.
The process can be divided into 3 conceptual main phases:

1. **Acceptance Phase**
   The transaction request must be created on first place, its origin must be verified to avoid fraud; once authenticated, the request is to be marked valid, encrypted and enqueued into the pending operations, awaiting miner's notice.

2. **Transmission Phase**
   Once the transaction is retrieved from the queue by the miner, the miner has to verify certain aspects of the transaction to be marked definitively acceptable (and therefore "voting" for valid) (Refer to "Section 4.3") by the miner and finally trace the receiver's node or the originating one, to which the vote is transmitted.

3. **Processing Phase**
   If a pending operation reached enough confirmations, the node can process its content and apply its actions permanently, registering the transaction into the blockchain's latest block available if not full, or create a new block where the transaction is put.

### 3.7.3 Miner Confirmation Process



The miner’s duty is to be considered fundamental for the verification process of a transaction and its actuation.

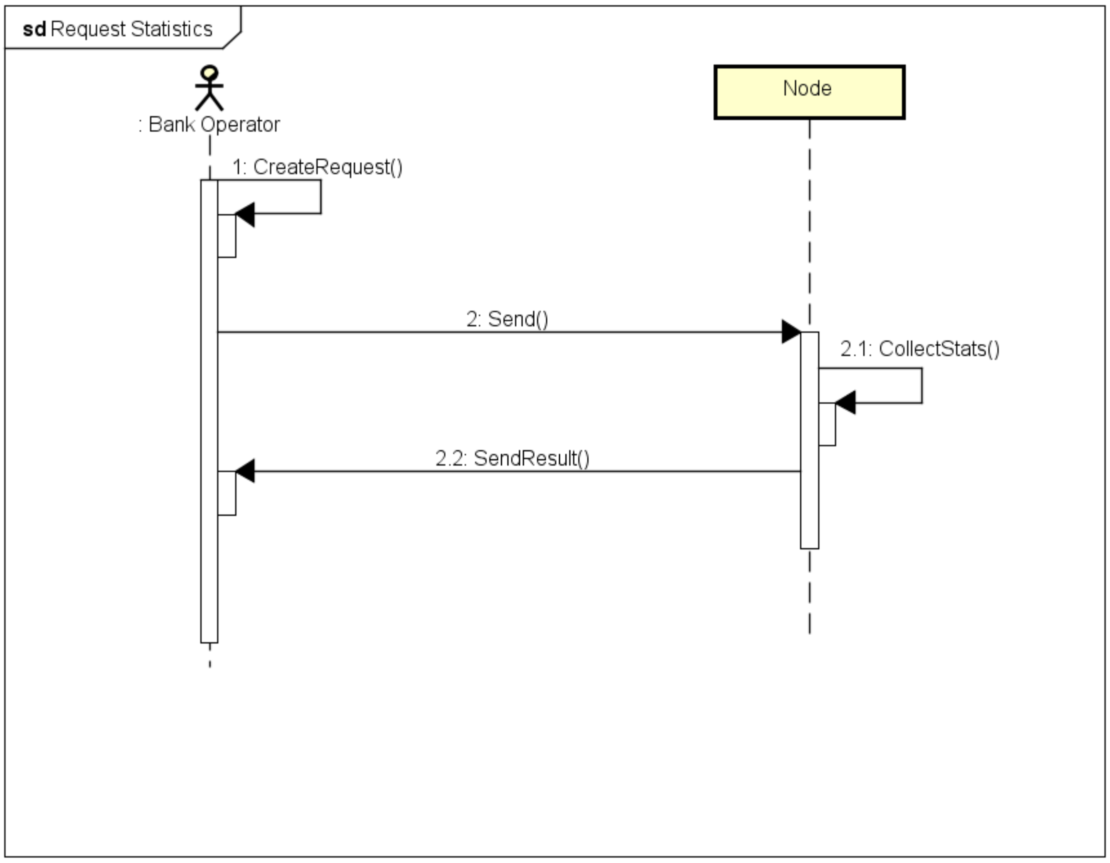
Once a transaction is retrieved by the pending operations, the transaction’s validity comes from the signatures included in the transaction request.

If the transaction signature is by the originating bank (and therefore the bank certified the origin of the transaction is by an authorized user), the miner must recover the receiver’s address and appropriate node(s) to send the vote to (in this project execution, only the one node is online).

The vote is also registered as an operation by implementing “IOperation” interface and will be registered inside the transaction “confirms”, in the block.

Whenever the transaction’s confirms reach the maximum necessary to be considered definitive, the transaction is automatically put into the latest available block and registered permanently.

### 3.7.4 Operator Statistics Request



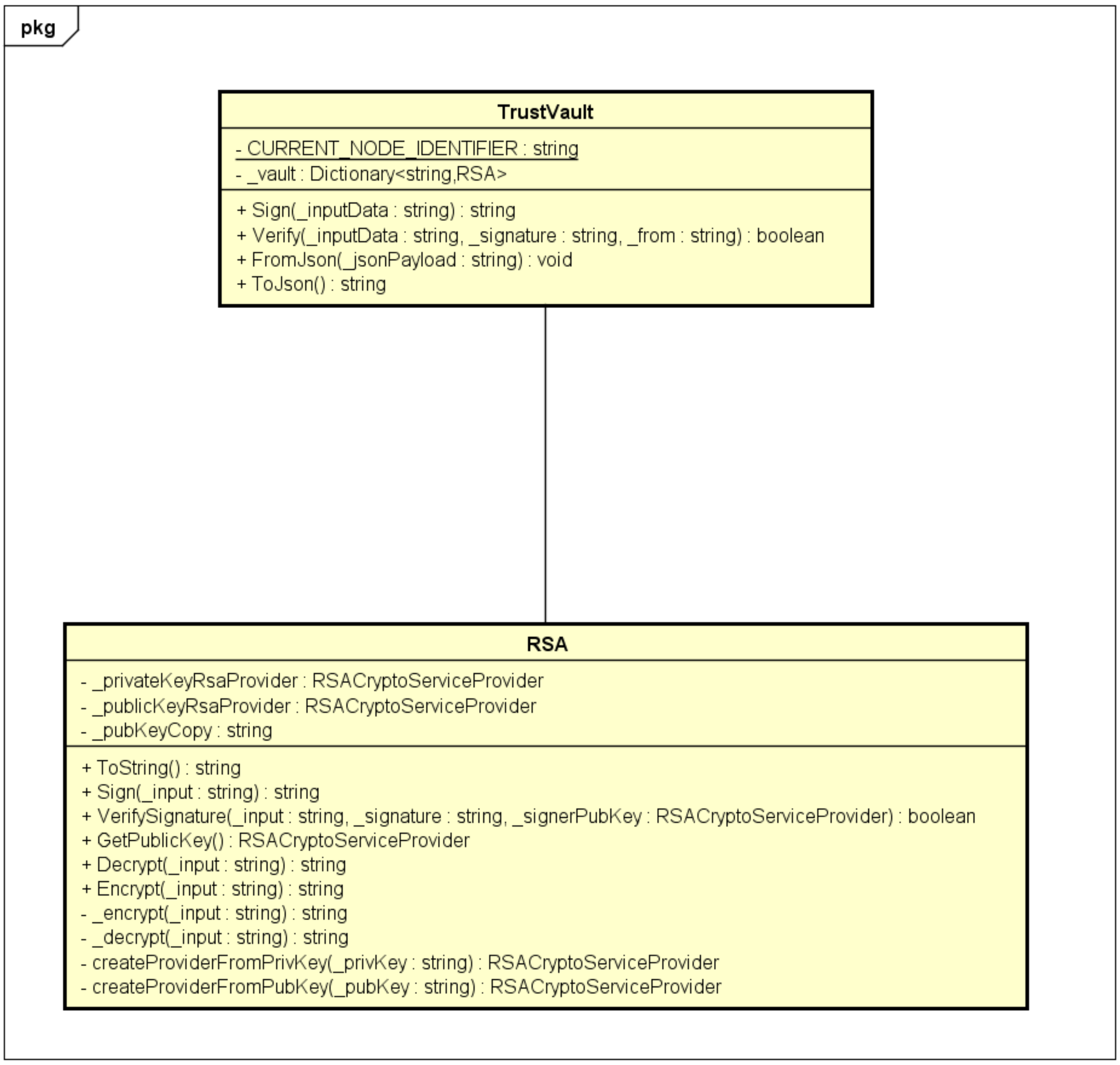
While the previously described operations may appear complicated, the statistics request is a simpler function used as a debug feature for the communication of the node and the content of the blockchain.
If the node recognizes authorization by the requesting's signature, the following statistics are sent back:

- Pending operations remaining
- Latest block information and content
  - Creation date
  - Containing transactions
  - Containing operations in general

### 3.7.5 Genesis Block

The genesis block is the first block created in the blockchain that will start the succession of the other blocks.
Its presence is essential as it ensure the second block (first real-usage block) has the necessary security measures (Hash) coherent from the start.
In this implementation of the Nordic Blockchain, the genesis block is automatically created with non-significant content.

In the project, the genesis block is the deposit of all the public keys of the nodes with relative owner, represented by a plain Block containing a single transaction, of which content is the “Trust Vault” structure.


pkg
TrustVault
- CURRENT_NODE_IDENTIFIER : string
- _vault : Dictionary<string,RSA>
+ Sign(_inputData : string) : string
+ Verify(_inputData : string, _signature : string, _from : string) : boolean
+ FromJson(_jsonPayload : string) : void
+ ToJson() : string
RSA
- _privateKeyRsaProvider : RSACryptoServiceProvider
- _publicKeyRsaProvider : RSACryptoServiceProvider
- _pubKeyCopy : string
+ ToString() : string
+ Sign(_input : string) : string
+ VerifySignature(_input : string, _signature : string, _signerPubKey : RSACryptoServiceProvider) : boolean
+ GetPublicKey() : RSACryptoServiceProvider
+ Decrypt(_input : string) : string
+ Encrypt(_input : string) : string
- _encrypt(_input : string) : string
- _decrypt(_input : string) : string
- createProviderFromPrivKey(_privKey : string) : RSACryptoServiceProvider
- createProviderFromPubKey(_pubKey : string) : RSACryptoServiceProvider

### 3.7.6 Overall architecture

The overall architecture shows the relationship between the components of the Nordic Blockchain project, both by “packages” (contextual sections) and the full version.
Refer to Appendix – Overall

## 3.8 Choice of technologies

### 3.8.1 Asymmetric Cryptography

The project and its security heavily rely on asymmetric cryptography to authenticate and encrypt the data received and sent, as well for authorization levels determining and origin verification.
While an alternative to RSA is ECDSA and is even newer technology, the choice of RSA in this project is purely for simplicity, compatibility coverage and accessibility as well as its maturity.

RSA is a well-known asymmetric cryptography algorithm, created in 1977 compared to ECDSA that has been proposed in 1992; thus, ECDSA is newer and its performance and time-complexity is supposedly better than RSA, despite that, RSA has better performance due to longer time of improvements.
While ECDSA offers better scaling capabilities and bigger key sizes, its vulnerability to Shor’s algorithm is considered weaker than RSA, that offers better resistance to quantum-based attacks, features longer maturity and a long list of adoptions in a huge variety of software, both FOSS and private.

For the above-mentioned reasons, the Nordic Blockchain’s team decided to adopt RSA for simplicity and extended documentation coverage compared to ECSA and a multitude of available APIs covered by different licenses, mainly FOSS-oriented.

### 3.8.2 WebSocket

The communication classes (Network, Client, SessionManager) use WebSockets for communication protocol instead of plain sockets.
The choice of WebSockets is to put a baseline for compatibility with multiple applications, including external ones, that desire to use such service, since the

WebSocket’s transportation protocol obeys under the HTTP standards and supports SSL.
Also, the usage of standardized protocol and technologies is always preferred over custom implementations that might deteriorate over time and decrease maintainability.

### 3.8.3 .NET Core

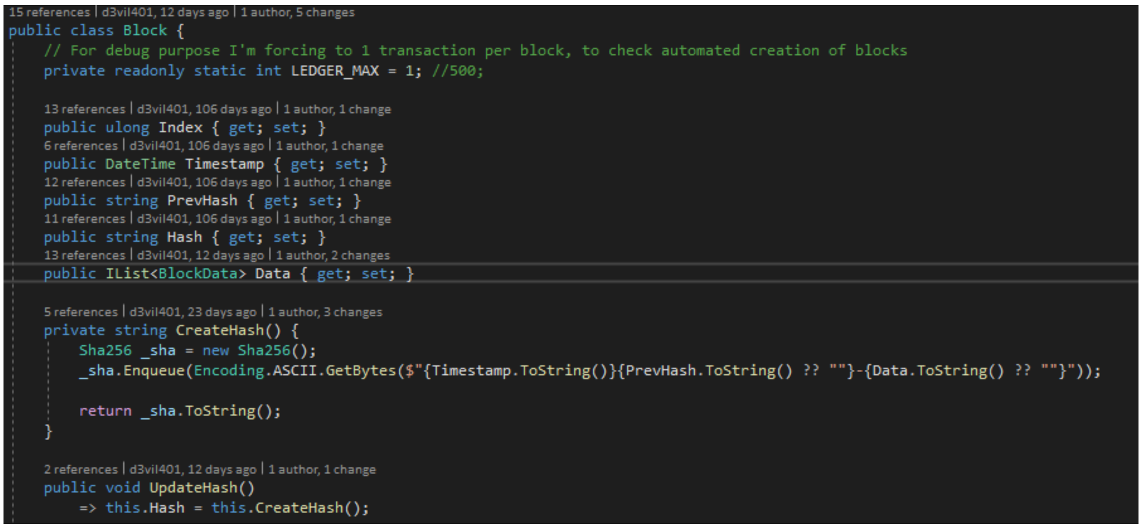
The choice of .NET Core technology instead of .NET Framework is simply for simplicity of the C# language and, majorly, for cross-compilation and cross-compatibility towards different operative systems.
C# is becoming an increasingly influential programming language every year and the recent porting of the .NET technology into Linux/Unix systems is nonetheless speeding up the process of propagation, the author considered the possibility of keeping a project like this on multiple systems, considering the banks networks may have a bias towards a specific OS.

### 3.8.4 NuGet Package Manager

The installation and usage of external libraries is possible thanks to the NuGet package manager.
The software is a one-click package installer for libraries developed by any entity (private or corporate) and it is included into Visual Studio IDE, speeding up the development times for the author of the project.
NuGet may not be suitable for certified-only production development – concerning security - and may retain further investigation for production quality and critical security based projects.

# 4 Implementation

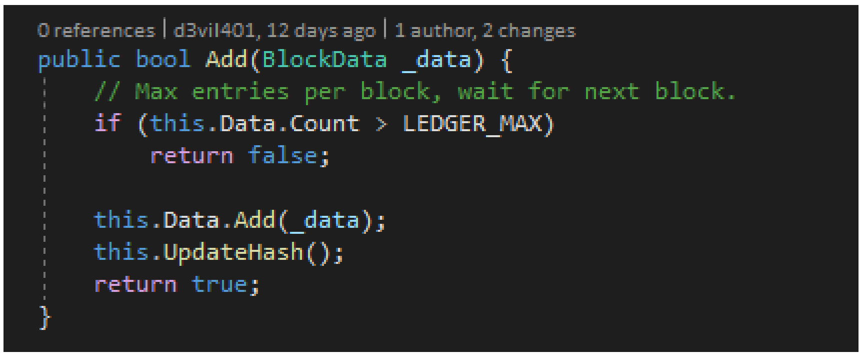
## 4.1 Block



```
15 references | d3vil401, 12 days ago | 1 author, 5 changes
public class Block {
    // For debug purpose I'm forcing to 1 transaction per block, to check automated creation of blocks
    private readonly static int LEDGER_MAX = 1; //500;

    13 references | d3vil401, 106 days ago | 1 author, 1 change
    public ulong Index { get; set; }
    6 references | d3vil401, 106 days ago | 1 author, 1 change
    public DateTime Timestamp { get; set; }
    12 references | d3vil401, 106 days ago | 1 author, 1 change
    public string PrevHash { get; set; }
    11 references | d3vil401, 106 days ago | 1 author, 1 change
    public string Hash { get; set; }
    13 references | d3vil401, 12 days ago | 1 author, 2 changes
    public IList<BlockData> Data { get; set; }

    5 references | d3vil401, 23 days ago | 1 author, 3 changes
    private string CreateHash() {
        Sha256 _sha = new Sha256();
        _sha.Enqueue(Encoding.ASCII.GetBytes($"{Timestamp.ToString()}{PrevHash.ToString() ?? ""}-{Data.ToString() ?? ""}"));

        return _sha.ToString();
    }

    2 references | d3vil401, 12 days ago | 1 author, 1 change
    public void UpdateHash()
        => this.Hash = this.CreateHash();
```

The block structure and calculation of hash.

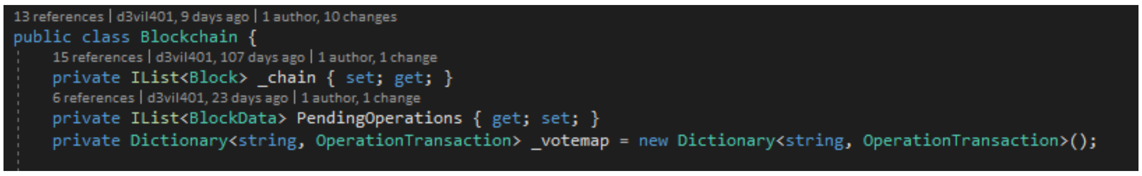
The hash calculation is a merging of the previous block hash, the creation time and the content, which ensures the concatenation of blocks that are recursively verified for compromise verification.



```
0 references | d3vil401, 12 days ago | 1 author, 2 changes
public bool Add(BlockData _data) {
    // Max entries per block, wait for next block.
    if (this.Data.Count > LEDGER_MAX)
        return false;

    this.Data.Add(_data);
    this.UpdateHash();
    return true;
}
```

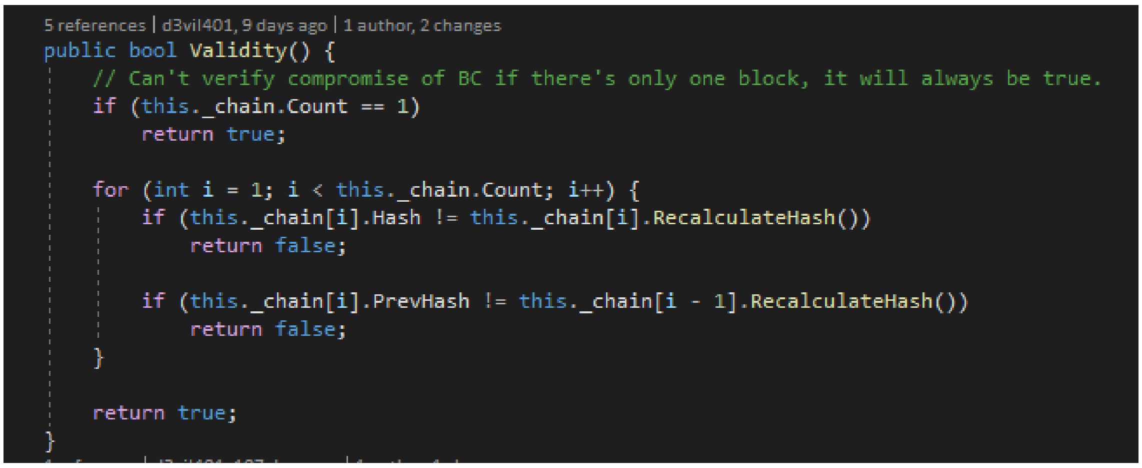
The function that registers the new data if not full and updates the current block hash whenever it changes.

## 4.2 Blockchain



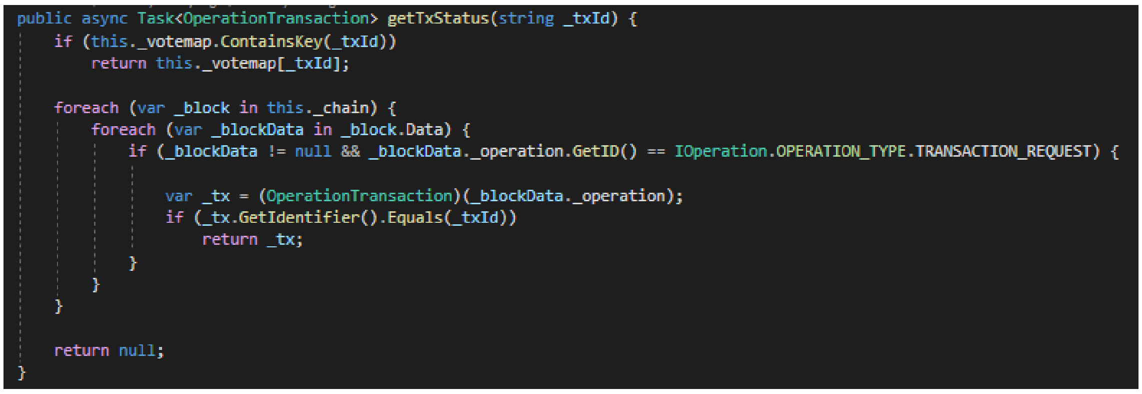
```
13 references | d3vil401, 9 days ago | 1 author, 10 changes
public class Blockchain {
    15 references | d3vil401, 107 days ago | 1 author, 1 change
    private IList<Block> _chain { set; get; }
    6 references | d3vil401, 23 days ago | 1 author, 1 change
    private IList<BlockData> PendingOperations { get; set; }
    private Dictionary<string, OperationTransaction> _votemap = new Dictionary<string, OperationTransaction>();
```

The blockchain contains 3 complex structures, the "_chain" is the current chain of blocks that contains permanently registered data, the "PendingOperations" holds the list of pending operations that the miners will request, this is updated automatically by removing newly confirmed operations and only adding newly created ones, and, the "_votemap" that holds the reference of transactions based on their Transaction Identifier, used for miner's vote casting.

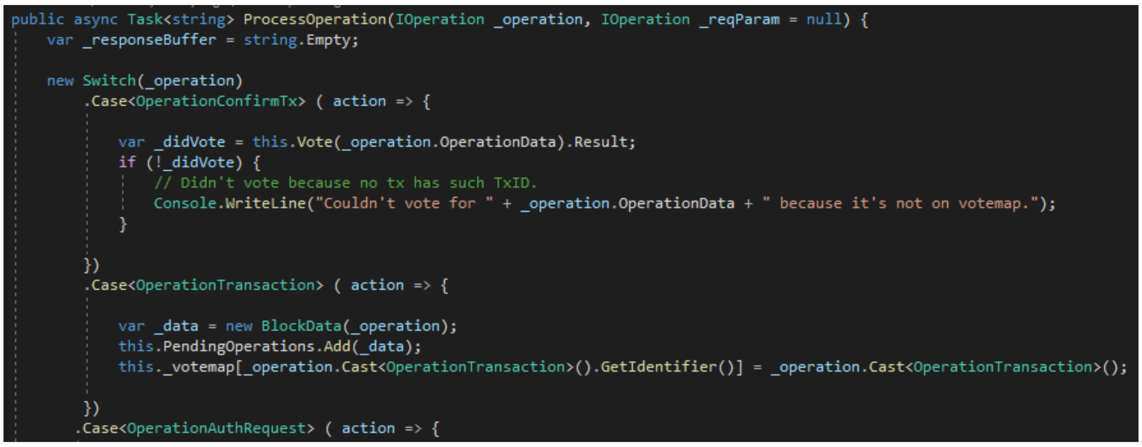


```
5 references | d3vil401, 9 days ago | 1 author, 2 changes
public bool Validity() {
    // Can't verify compromise of BC if there's only one block, it will always be true.
    if (this._chain.Count == 1)
        return true;

    for (int i = 1; i < this._chain.Count; i++) {
        if (this._chain[i].Hash != this._chain[i].RecalculateHash())
            return false;

        if (this._chain[i].PrevHash != this._chain[i - 1].RecalculateHash())
            return false;
    }

    return true;
}
```

The "Validity" function is for verification of compromised of a blockchain, it compares the registered hashes between concatenated blocks and the latest hash version. Instead of relying on already registered hashes inside the block, the function forces the recalculation of these and compares that the registered hash is the same as the recalculated one.

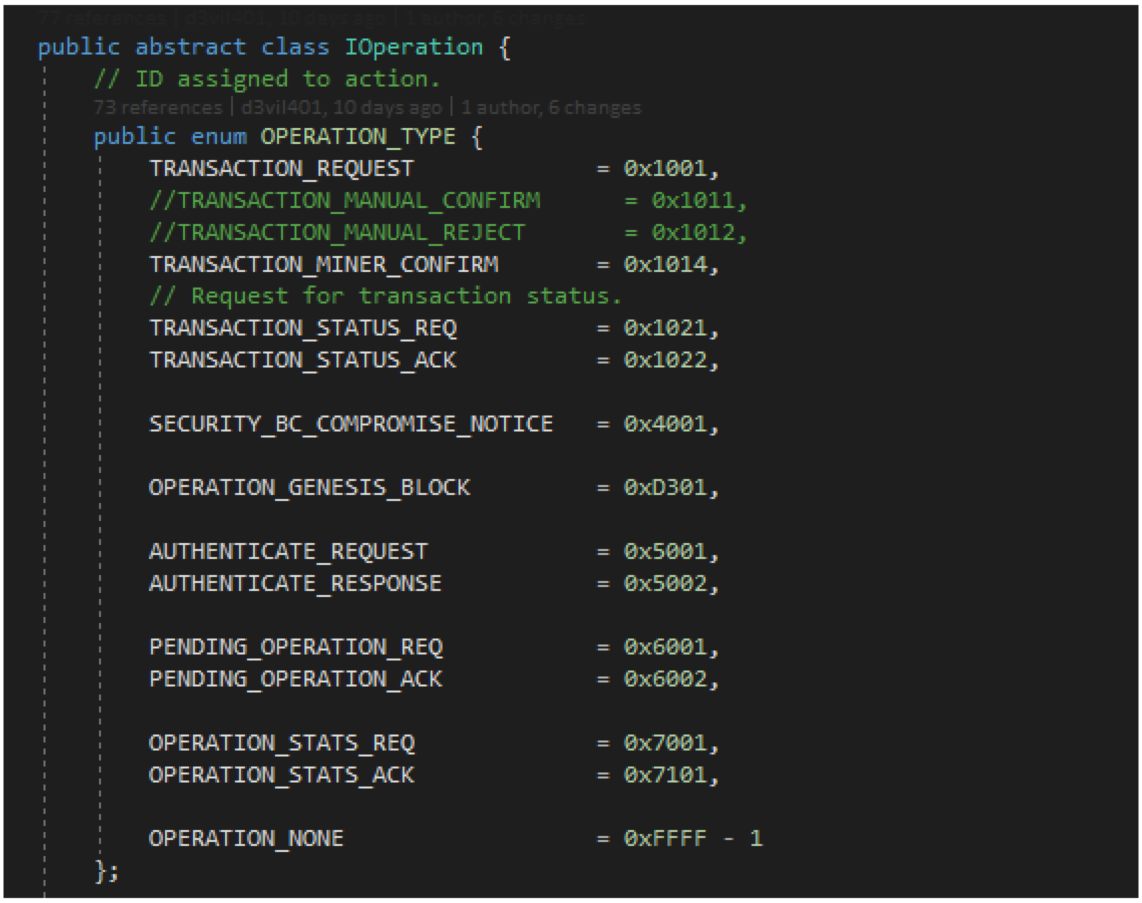

```
public async Task<OperationTransaction> getTxStatus(string _txId) {
    if (this._votemap.ContainsKey(_txId))
        return this._votemap[_txId];

    foreach (var _block in this._chain) {
        foreach (var _blockData in _block.Data) {
            if (_blockData != null && _blockData._operation.GetID() == IOperation.OPERATION_TYPE.TRANSACTION_REQUEST) {

                var _tx = (OperationTransaction)(_blockData._operation);
                if (_tx.GetIdentifier().Equals(_txId))
                    return _tx;
            }
        }
    }

    return null;
}
```

The “getTxStatus” method is used to get the transaction status by its identifier, the transaction is first being searched in the pending operations if it hasn’t been confirmed; if it has not been found in the vote map, the transaction may have been confirmed already and therefore must be searched in the various blocks registered in the blockchain.


```
public async Task<string> ProcessOperation(IOperation _operation, IOperation _reqParam = null) {
    var _responseBuffer = string.Empty;

    new Switch(_operation)
        .Case<OperationConfirmTx> ( action => {

            var _didVote = this.Vote(_operation.OperationData).Result;
            if (!_didVote) {
                // Didn't vote because no tx has such TxID.
                Console.WriteLine("Couldn't vote for " + _operation.OperationData + " because it's not on votemap.");
            }

        })
        .Case<OperationTransaction> ( action => {

            var _data = new BlockData(_operation);
            this.PendingOperations.Add(_data);
            this._votemap[_operation.Cast<OperationTransaction>().GetIdentifier()] = _operation.Cast<OperationTransaction>();

        })
       .Case<OperationAuthRequest> ( action => {
```

The “ProcessOperation” is the call-back function used by the network for operations that should interact with the blockchain or its content.
It functions as a bridge of communication between networking and blockchain manipulation that is under supervision of the developer’s decision of handling.
Based on a custom “Switch” by class type, the function handles the required operation and returns a CLM buffer if the server should respond to the client.

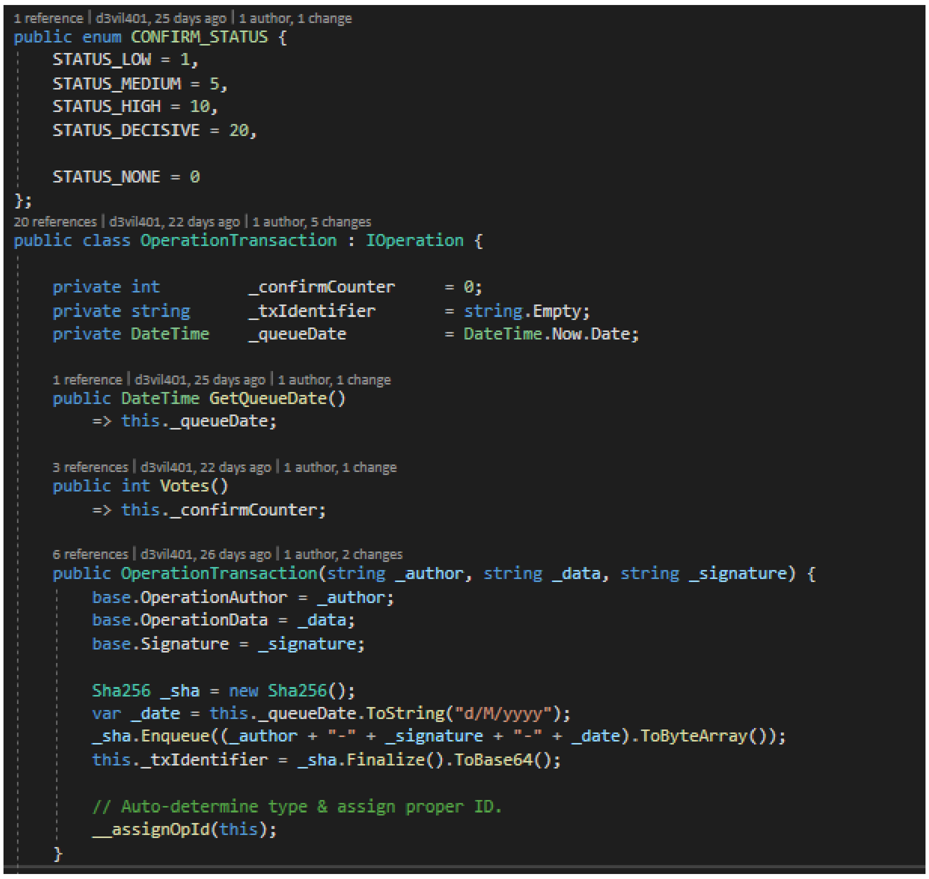
## 4.3 Operations



```
77 references | d3vil401, 10 days ago | 1 author, 6 changes
public abstract class IOperation {
    // ID assigned to action.
    73 references | d3vil401, 10 days ago | 1 author, 6 changes
    public enum OPERATION_TYPE {
        TRANSACTION_REQUEST                = 0x1001,
        //TRANSACTION_MANUAL_CONFIRM       = 0x1011,
        //TRANSACTION_MANUAL_REJECT        = 0x1012,
        TRANSACTION_MINER_CONFIRM          = 0x1014,
        // Request for transaction status.
        TRANSACTION_STATUS_REQ             = 0x1021,
        TRANSACTION_STATUS_ACK             = 0x1022,

        SECURITY_BC_COMPROMISE_NOTICE      = 0x4001,

        OPERATION_GENESIS_BLOCK            = 0xD301,

        AUTHENTICATE_REQUEST               = 0x5001,
        AUTHENTICATE_RESPONSE              = 0x5002,

        PENDING_OPERATION_REQ              = 0x6001,
        PENDING_OPERATION_ACK              = 0x6002,

        OPERATION_STATS_REQ                = 0x7001,
        OPERATION_STATS_ACK                = 0x7101,

        OPERATION_NONE                     = 0xFFFF - 1
    };
```

The registered IDs for data transmission, differentiated to ensure uniqueness and proper identification of the operation request.
It is fundamental that the operation ID is unique for every single packet since the CLM distinguishes the packets based on this.

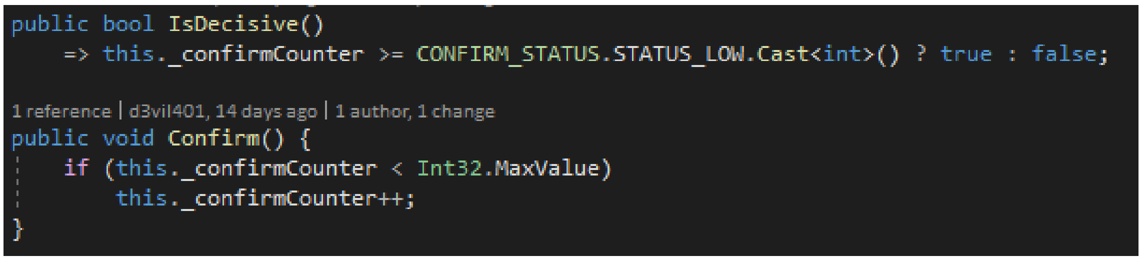
### 4.3.1 Transaction Operation



```
1 reference | d3vil401, 25 days ago | 1 author, 1 change
public enum CONFIRM_STATUS {
    STATUS_LOW = 1,
    STATUS_MEDIUM = 5,
    STATUS_HIGH = 10,
    STATUS_DECISIVE = 20,

    STATUS_NONE = 0
};
20 references | d3vil401, 22 days ago | 1 author, 5 changes
public class OperationTransaction : IOperation {

    private int         _confirmCounter     = 0;
    private string      _txIdentifier       = string.Empty;
    private DateTime    _queueDate          = DateTime.Now.Date;

    1 reference | d3vil401, 25 days ago | 1 author, 1 change
    public DateTime GetQueueDate()
        => this._queueDate;

    3 references | d3vil401, 22 days ago | 1 author, 1 change
    public int Votes()
        => this._confirmCounter;

    6 references | d3vil401, 26 days ago | 1 author, 2 changes
    public OperationTransaction(string _author, string _data, string _signature) {
        base.OperationAuthor = _author;
        base.OperationData = _data;
        base.Signature = _signature;

        Sha256 _sha = new Sha256();
        var _date = this._queueDate.ToString("d/M/yyyy");
        _sha.Enqueue((_author + "-" + _signature + "-" + _date).ToByteArray());
        this._txIdentifier = _sha.Finalize().ToBase64();

        // Auto-determine type & assign proper ID.
        __assignOpId(this);
    }
```

The transaction structure and assignation of unique transaction ID.

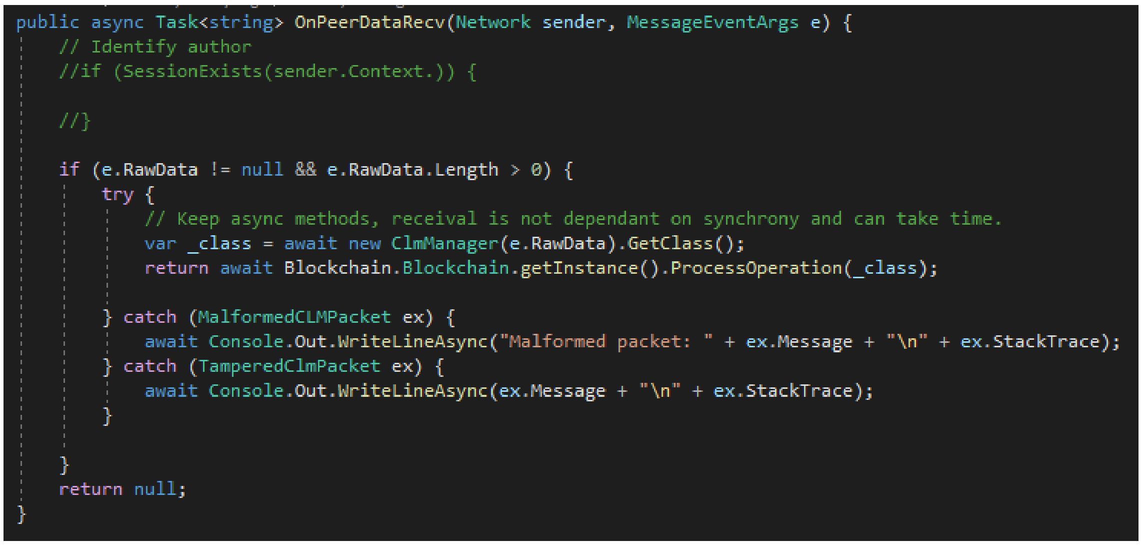
To ensure the transaction ID is unique for every transaction but also the chance for the transaction to be re-traceable in case a user lost the TxID, the information the client must provide for his/her identity combined with the time of request will output the TxID.



```
public bool IsDecisive()
    => this._confirmCounter >= CONFIRM_STATUS.STATUS_LOW.Cast<int>() ? true : false;

1 reference | d3vil401, 14 days ago | 1 author, 1 change
public void Confirm() {
    if (this._confirmCounter < Int32.MaxValue)
        this._confirmCounter++;
}
```

Decisive status of transaction function and vote casting function, respectively.

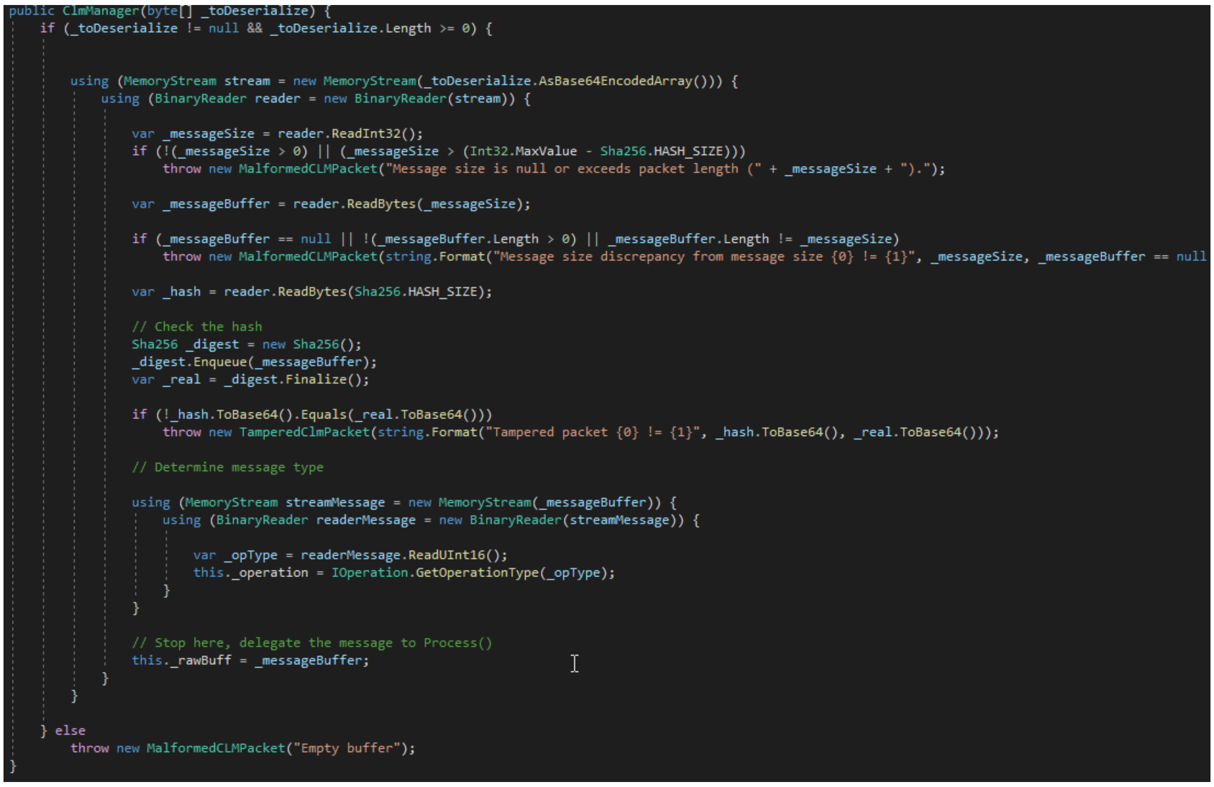
## 4.4 Network



```
public async Task<string> OnPeerDataRecv(Network sender, MessageEventArgs e) {
    // Identify author
    //if (SessionExists(sender.Context.)) {

    //}

    if (e.RawData != null && e.RawData.Length > 0) {
        try {
            // Keep async methods, receival is not dependant on synchrony and can take time.
            var _class = await new ClmManager(e.RawData).GetClass();
            return await Blockchain.Blockchain.getInstance().ProcessOperation(_class);

        } catch (MalformedCLMPacket ex) {
            await Console.Out.WriteLineAsync("Malformed packet: " + ex.Message + "\n" + ex.StackTrace);
        } catch (TamperedClmPacket ex) {
            await Console.Out.WriteLineAsync(ex.Message + "\n" + ex.StackTrace);
        }

    }
    return null;
}
```

The first entry point of received data from the WebSocket, being parsed trough the “CLM Manager” for protocol handling and then processed.

This is the entry point of all of the communications received, coordinating the various operations.



```csharp
public ClmManager(byte[] _toDeserialize) {
    if (_toDeserialize != null && _toDeserialize.Length >= 0) {


        using (MemoryStream stream = new MemoryStream(_toDeserialize.AsBase64EncodedArray())) {
            using (BinaryReader reader = new BinaryReader(stream)) {

                var _messageSize = reader.ReadInt32();
                if (!(_messageSize > 0) || (_messageSize > (Int32.MaxValue - Sha256.HASH_SIZE)))
                    throw new MalformedCLMPacket("Message size is null or exceeds packet length (" + _messageSize + ").");

                var _messageBuffer = reader.ReadBytes(_messageSize);

                if (_messageBuffer == null || !(_messageBuffer.Length > 0) || _messageBuffer.Length != _messageSize)
                    throw new MalformedCLMPacket(string.Format("Message size discrepancy from message size {0} != {1}", _messageSize, _messageBuffer == null

                var _hash = reader.ReadBytes(Sha256.HASH_SIZE);

                // Check the hash
                Sha256 _digest = new Sha256();
                _digest.Enqueue(_messageBuffer);
                var _real = _digest.Finalize();

                if (!_hash.ToBase64().Equals(_real.ToBase64()))
                    throw new TamperedClmPacket(string.Format("Tampered packet {0} != {1}", _hash.ToBase64(), _real.ToBase64()));

                // Determine message type

                using (MemoryStream streamMessage = new MemoryStream(_messageBuffer)) {
                    using (BinaryReader readerMessage = new BinaryReader(streamMessage)) {

                        var _opType = readerMessage.ReadUInt16();
                        this._operation = IOperation.GetOperationType(_opType);
                    }
                }

                // Stop here, delegate the message to Process()
                this._rawBuff = _messageBuffer;
            }
        }

    } else
        throw new MalformedCLMPacket("Empty buffer");
}
```

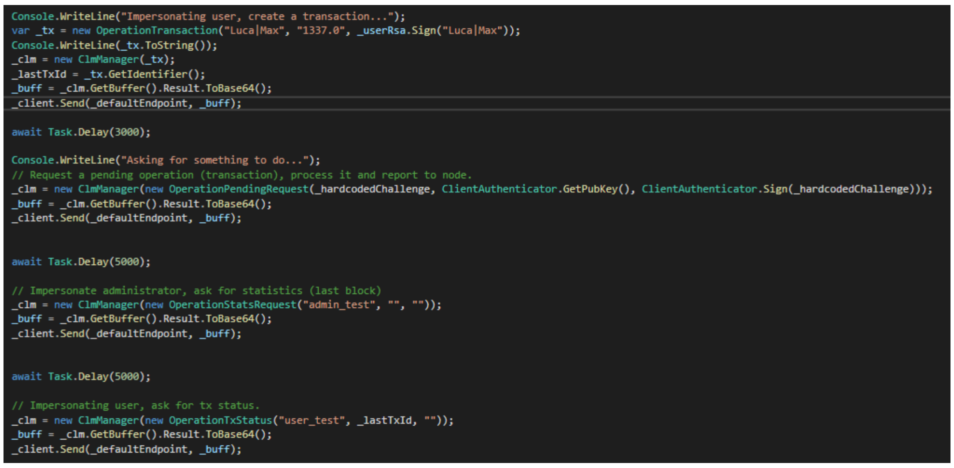
The encapsulation processing function, verifying the tampering as well.

The CLM does not handle yet the deserialization of the packet back to the class, this function only takes care of encapsulation and the validity of it, the deserialization is handled right after the execution of this method.

## 4.5 Test Miner

The testing of the miner trough unit-tests was not recommended since it requires specific manual checks, forcing the tester to solely rely on debugging and manual verification.

To allow so, a specific sub-project is created (“NordicMiner”) with the sole purpose of being both and example of a miner for future developers approaching the project and as the testing platform for the mining functionalities.

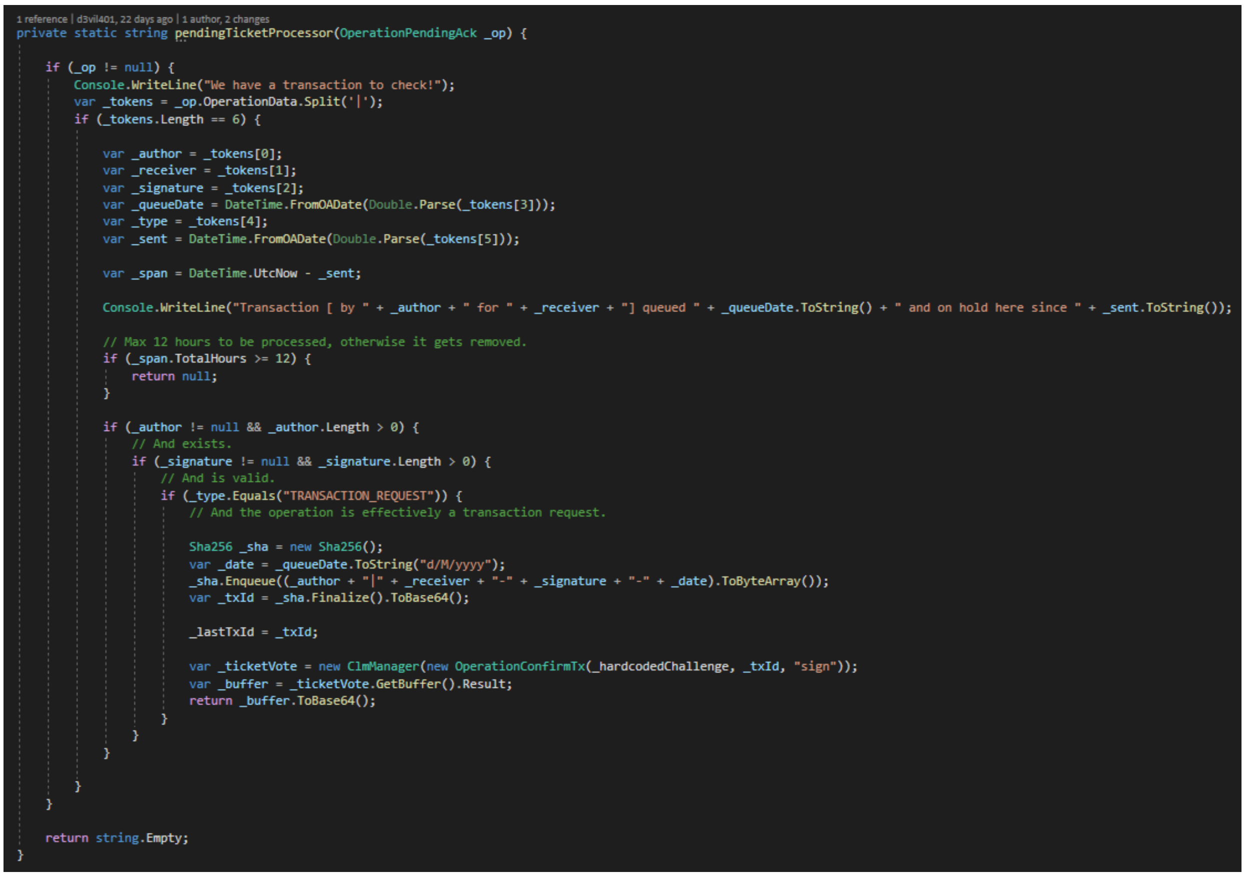


```
Console.WriteLine("Impersonating user, create a transaction...");
var _tx = new OperationTransaction("Luca|Max", "1337.0", _userRsa.Sign("Luca|Max"));
Console.WriteLine(_tx.ToString());
_clm = new ClmManager(_tx);
_lastTxId = _tx.GetIdentifier();
_buff = _clm.GetBuffer().Result.ToBase64();
_client.Send(_defaultEndpoint, _buff);

await Task.Delay(3000);

Console.WriteLine("Asking for something to do...");
// Request a pending operation (transaction), process it and report to node.
_clm = new ClmManager(new OperationPendingRequest(_hardcodedChallenge, ClientAuthenticator.GetPubKey(), ClientAuthenticator.Sign(_hardcodedChallenge)));
_buff = _clm.GetBuffer().Result.ToBase64();
_client.Send(_defaultEndpoint, _buff);


await Task.Delay(5000);

// Impersonate administrator, ask for statistics (last block)
_clm = new ClmManager(new OperationStatsRequest("admin_test", "", ""));
_buff = _clm.GetBuffer().Result.ToBase64();
_client.Send(_defaultEndpoint, _buff);


await Task.Delay(5000);

// Impersonating user, ask for tx status.
_clm = new ClmManager(new OperationTxStatus("user_test", _lastTxId, ""));
_buff = _clm.GetBuffer().Result.ToBase64();
_client.Send(_defaultEndpoint, _buff);
```

The testing miner, impersonating respectively:

1. The user role, creating a transaction request.
2. The miner role, asking and processing a pending operation.
3. The admin role, asking and printing the node's statistics.
4. The user role, asking for the previously confirmed and registered transaction by using the previously obtained transaction identifier.

The delays are pseudo network delays and allows the node to respond to the miner.

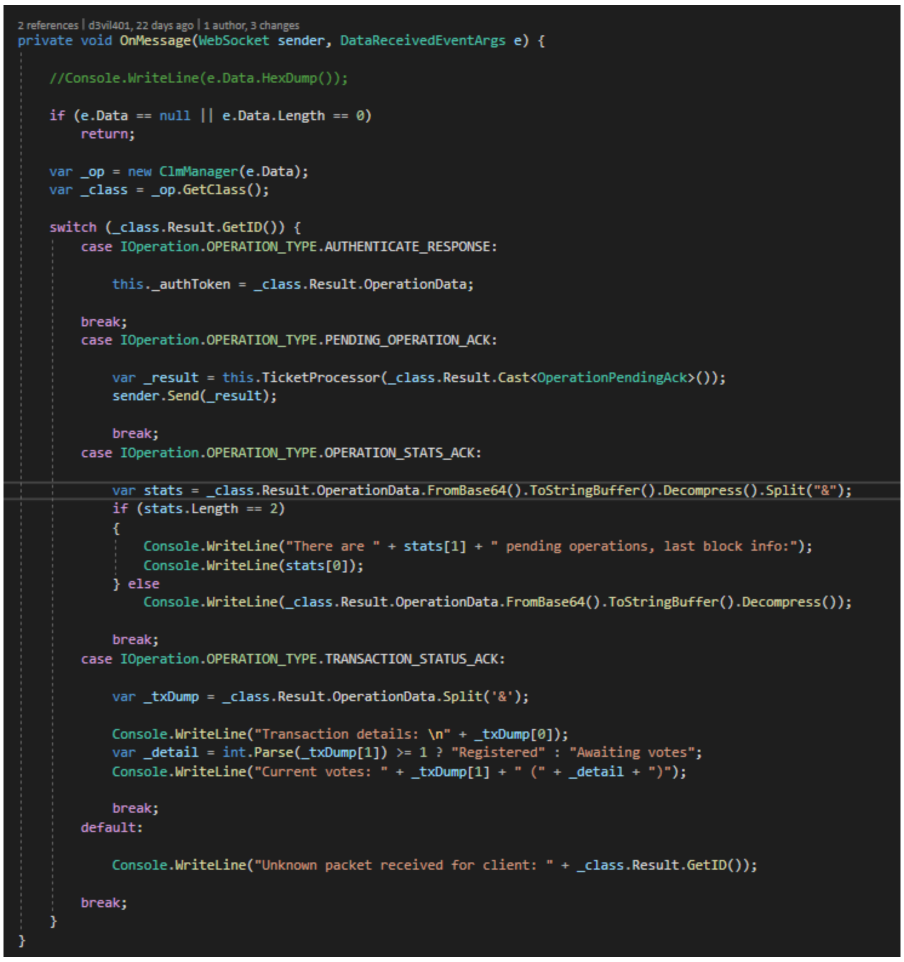


```
1 reference | d3vil401, 22 days ago | 1 author, 2 changes
private static string pendingTicketProcessor(OperationPendingAck _op) {

    if (_op != null) {
        Console.WriteLine("We have a transaction to check!");
        var _tokens = _op.OperationData.Split('|');
        if (_tokens.Length == 6) {

            var _author = _tokens[0];
            var _receiver = _tokens[1];
            var _signature = _tokens[2];
            var _queueDate = DateTime.FromOADate(Double.Parse(_tokens[3]));
            var _type = _tokens[4];
            var _sent = DateTime.FromOADate(Double.Parse(_tokens[5]));

            var _span = DateTime.UtcNow - _sent;

            Console.WriteLine("Transaction [ by " + _author + " for " + _receiver + "] queued " + _queueDate.ToString() + " and on hold here since " + _sent.ToString());

            // Max 12 hours to be processed, otherwise it gets removed.
            if (_span.TotalHours >= 12) {
                return null;
            }

            if (_author != null && _author.Length > 0) {
                // And exists.
                if (_signature != null && _signature.Length > 0) {
                    // And is valid.
                    if (_type.Equals("TRANSACTION_REQUEST")) {
                        // And the operation is effectively a transaction request.

                        Sha256 _sha = new Sha256();
                        var _date = _queueDate.ToString("d/M/yyyy");
                        _sha.Enqueue((_author + "|" + _receiver + "-" + _signature + "-" + _date).ToByteArray());
                        var _txId = _sha.Finalize().ToBase64();

                        _lastTxId = _txId;

                        var _ticketVote = new ClmManager(new OperationConfirmTx(_hardcodedChallenge, _txId, "sign"));
                        var _buffer = _ticketVote.GetBuffer().Result;
                        return _buffer.ToBase64();
                    }
                }
            }

        }
    }

    return string.Empty;
}
```

This is the testing function for a transaction to be checked.

The information in the “OperationPendingAck” are joined trough a ‘|’ character, allowing the method to extract the information required for the verification of the transaction.

The TxID is not exposed from the “OperationPendingAck”, ensuring that if the received information is correct the TxID calculated from the miner is valid and the node can record the vote casted from the miner.



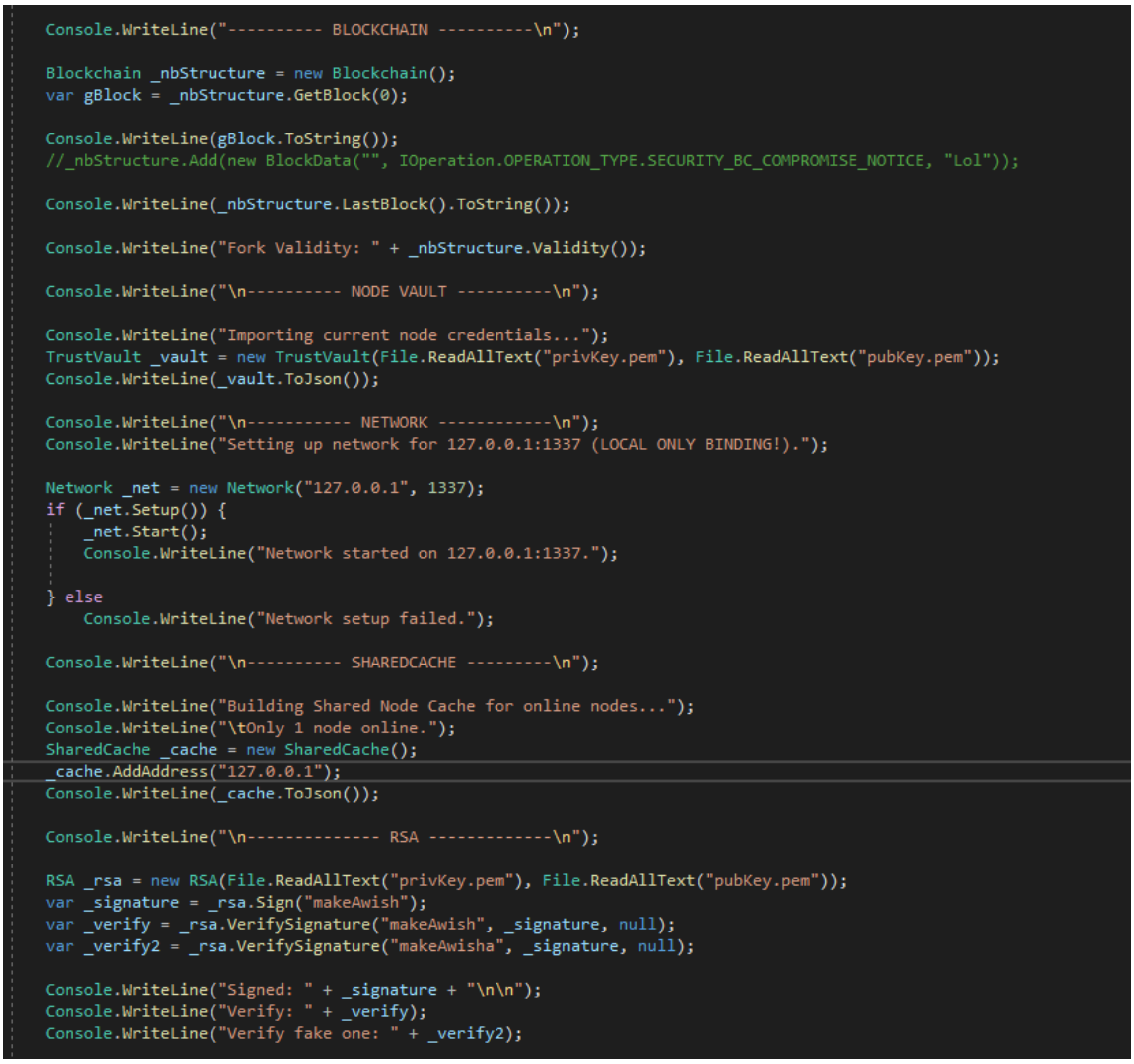
```
2 references | d3vil401, 22 days ago | 1 author, 3 changes
private void OnMessage(WebSocket sender, DataReceivedEventArgs e) {

    //Console.WriteLine(e.Data.HexDump());

    if (e.Data == null || e.Data.Length == 0)
        return;

    var _op = new ClmManager(e.Data);
    var _class = _op.GetClass();

    switch (_class.Result.GetID()) {
        case IOperation.OPERATION_TYPE.AUTHENTICATE_RESPONSE:

            this._authToken = _class.Result.OperationData;

        break;
        case IOperation.OPERATION_TYPE.PENDING_OPERATION_ACK:

            var _result = this.TicketProcessor(_class.Result.Cast<OperationPendingAck>());
            sender.Send(_result);

            break;
        case IOperation.OPERATION_TYPE.OPERATION_STATS_ACK:

            var stats = _class.Result.OperationData.FromBase64().ToStringBuffer().Decompress().Split("&");
            if (stats.Length == 2)
            {
                Console.WriteLine("There are " + stats[1] + " pending operations, last block info:");
                Console.WriteLine(stats[0]);
            } else
                Console.WriteLine(_class.Result.OperationData.FromBase64().ToStringBuffer().Decompress());

            break;
        case IOperation.OPERATION_TYPE.TRANSACTION_STATUS_ACK:

            var _txDump = _class.Result.OperationData.Split('&');

            Console.WriteLine("Transaction details: \n" + _txDump[0]);
            var _detail = int.Parse(_txDump[1]) >= 1 ? "Registered" : "Awaiting votes";
            Console.WriteLine("Current votes: " + _txDump[1] + " (" + _detail + ")");

            break;
        default:

            Console.WriteLine("Unknown packet received for client: " + _class.Result.GetID());

        break;
    }
}
```

The Client’s “OnMessage” method handles the responses to the operations from the node and the extraction and visualization of the information received by the Client. Here can be seen the 3 major functionalities of the client, respectively

1. The redirection of the received pending operation to the processor function previously described.
2. The receival of statistics from the node, after the miner impersonated the administrator and requested the statistics; the statistics received are the latest block’s dump and the number of pending operations awaiting processing.
3. The transaction status request, after the miner impersonated the user and requested the status of the transaction; the status vary from the implementation, in this prototype only 1 vote is required to register the transaction in the blockchain.

## 4.6 Node Test

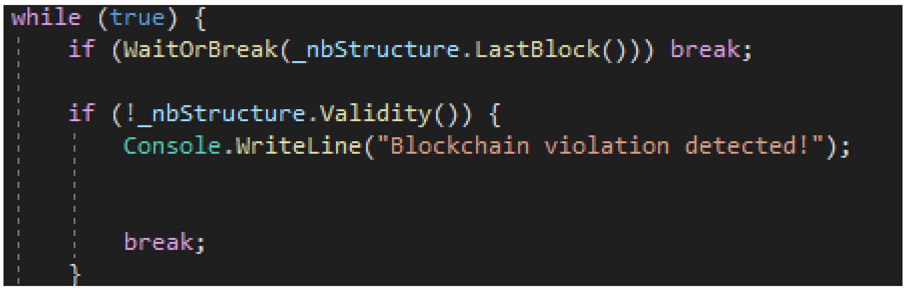
Similarly, to the miner test situation, the node must be tested accordingly with an ad-hoc project for manual debugging and stepping trough the code; additionally, the “NBService” project offers an example of a node setup.



```
Console.WriteLine("---------- BLOCKCHAIN ----------\n");

Blockchain _nbStructure = new Blockchain();
var gBlock = _nbStructure.GetBlock(0);

Console.WriteLine(gBlock.ToString());
//_nbStructure.Add(new BlockData("", IOperation.OPERATION_TYPE.SECURITY_BC_COMPROMISE_NOTICE, "Lol"));

Console.WriteLine(_nbStructure.LastBlock().ToString());

Console.WriteLine("Fork Validity: " + _nbStructure.Validity());

Console.WriteLine("\n---------- NODE VAULT ----------\n");

Console.WriteLine("Importing current node credentials...");
TrustVault _vault = new TrustVault(File.ReadAllText("privKey.pem"), File.ReadAllText("pubKey.pem"));
Console.WriteLine(_vault.ToJson());

Console.WriteLine("\n----------- NETWORK ------------\n");
Console.WriteLine("Setting up network for 127.0.0.1:1337 (LOCAL ONLY BINDING!).");

Network _net = new Network("127.0.0.1", 1337);
if (_net.Setup()) {
    _net.Start();
    Console.WriteLine("Network started on 127.0.0.1:1337.");

} else
    Console.WriteLine("Network setup failed.");

Console.WriteLine("\n---------- SHAREDCACHE ---------\n");

Console.WriteLine("Building Shared Node Cache for online nodes...");
Console.WriteLine("\tOnly 1 node online.");
SharedCache _cache = new SharedCache();
_cache.AddAddress("127.0.0.1");
Console.WriteLine(_cache.ToJson());

Console.WriteLine("\n-------------- RSA -------------\n");

RSA _rsa = new RSA(File.ReadAllText("privKey.pem"), File.ReadAllText("pubKey.pem"));
var _signature = _rsa.Sign("makeAwish");
var _verify = _rsa.VerifySignature("makeAwish", _signature, null);
var _verify2 = _rsa.VerifySignature("makeAwisha", _signature, null);

Console.WriteLine("Signed: " + _signature + "\n\n");
Console.WriteLine("Verify: " + _verify);
Console.WriteLine("Verify fake one: " + _verify2);
```

Here can be seen how the node first sets up the blockchain and performs some output on the console and verifies that the setup is coherent with the blockchain’s security standard (fork validity).

The credentials are then imported, the network is set up and the RSA functionalities are tested: this is the set-up checks performed before starting the service.



```
while (true) {
    if (WaitOrBreak(_nbStructure.LastBlock())) break;

    if (!_nbStructure.Validity()) {
        Console.WriteLine("Blockchain violation detected!");


        break;
    }
```

To avoid interruption of the program a loop is forcing the program to keep running until manually interrupted.

Additionally, the loop continuously checks the validity of the blockchain to a real-time compromise notice.

# 5 Test

This section describes employed testing methodologies, purpose and results.

### 5.1.1 Test Results

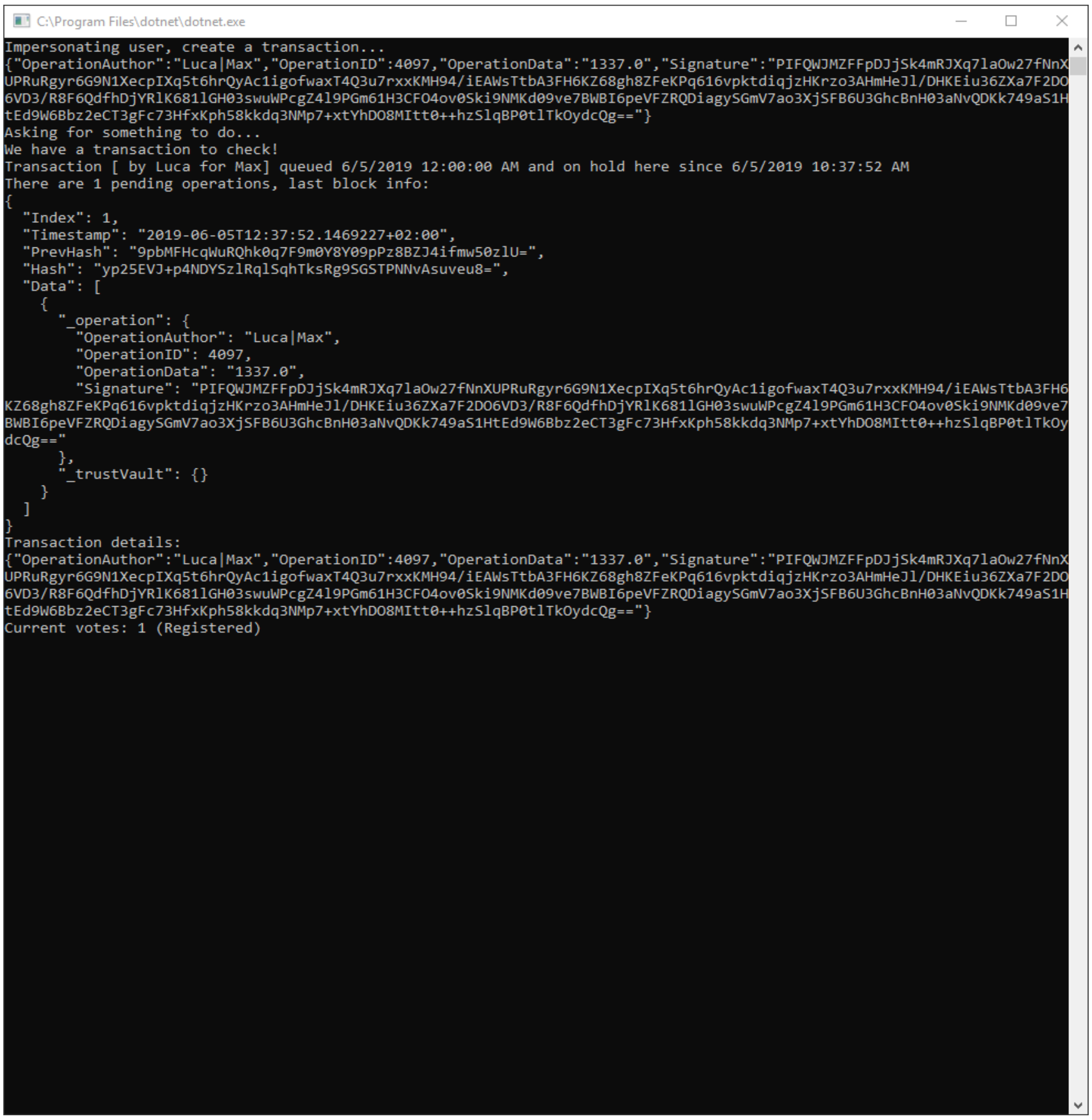
To verify implementation of the use cases previously described and to ensure stability of the proof of concept project, the tests are divided into 3 main parts:

- **Node Test –** The ad-hoc testing application for the node, it sets up the blockchain and checks its validity, tests the cryptographic components and the network and finally running the node service.
  The node test is to be analysed manually trough debugging to check the control flow of the code and data, but if the service is running then the tests are considerable positive.

- **Miner Test –** The miner ad-hoc testing application, it collects pending operations and confirms transactions (Use case 3), impersonates the administration for the collection of the statistics (Use case 2) and finally impersonates the user to check a transaction status (Use case 1 – 2).

- **Unit Tests –** Unit-tests are used to test the single components individually and regardless of their role inside the whole project; it is necessary to test that the single components are working as intended before the tests for their roles linked together commence.

#### 5.1.1.1 Miner Test


C:\Program Files\dotnet\dotnet.exe

```
Impersonating user, create a transaction...
{"OperationAuthor":"Luca|Max","OperationID":4097,"OperationData":"1337.0","Signature":"PIFQWJMZFFpDJjSk4mRJXq7laOw27fNnX
UPRuRgyr6G9N1XecpIXq5t6hrQyAc1igofwaxT4Q3u7rxxKMH94/iEAWsTtbA3FH6KZ68gh8ZFeKPq616vpktdiqjzHKrzo3AHmHeJl/DHKEiu36ZXa7F2DO
6VD3/R8F6QdfhDjYRlK681lGH03swuWPcgZ4l9PGm61H3CFO4ov0Ski9NMKd09ve7BWBI6peVFZRQDiagySGmV7ao3XjSFB6U3GhcBnH03aNvQDKk749aS1H
tEd9W6Bbz2eCT3gFc73HfxKph58kkdq3NMp7+xtYhDO8MItt0++hzSlqBP0tlTkOydcQg=="}
Asking for something to do...
We have a transaction to check!
Transaction [ by Luca for Max] queued 6/5/2019 12:00:00 AM and on hold here since 6/5/2019 10:37:52 AM
There are 1 pending operations, last block info:
{
  "Index": 1,
  "Timestamp": "2019-06-05T12:37:52.1469227+02:00",
  "PrevHash": "9pbMFHcqWuRQhk0q7F9m0Y8Y09pPz8BZJ4ifmw50zlU=",
  "Hash": "yp25EVJ+p4NDYSzlRqlSqhTksRg9SGSTPNNvAsuveu8=",
  "Data": [
    {
      "_operation": {
        "OperationAuthor": "Luca|Max",
        "OperationID": 4097,
        "OperationData": "1337.0",
        "Signature": "PIFQWJMZFFpDJjSk4mRJXq7laOw27fNnXUPRuRgyr6G9N1XecpIXq5t6hrQyAc1igofwaxT4Q3u7rxxKMH94/iEAWsTtbA3FH6
KZ68gh8ZFeKPq616vpktdiqjzHKrzo3AHmHeJl/DHKEiu36ZXa7F2DO6VD3/R8F6QdfhDjYRlK681lGH03swuWPcgZ4l9PGm61H3CFO4ov0Ski9NMKd09ve7
BWBI6peVFZRQDiagySGmV7ao3XjSFB6U3GhcBnH03aNvQDKk749aS1HtEd9W6Bbz2eCT3gFc73HfxKph58kkdq3NMp7+xtYhDO8MItt0++hzSlqBP0tlTkOy
dcQg=="
      },
      "_trustVault": {}
    }
  ]
}
Transaction details:
{"OperationAuthor":"Luca|Max","OperationID":4097,"OperationData":"1337.0","Signature":"PIFQWJMZFFpDJjSk4mRJXq7laOw27fNnX
UPRuRgyr6G9N1XecpIXq5t6hrQyAc1igofwaxT4Q3u7rxxKMH94/iEAWsTtbA3FH6KZ68gh8ZFeKPq616vpktdiqjzHKrzo3AHmHeJl/DHKEiu36ZXa7F2DO
6VD3/R8F6QdfhDjYRlK681lGH03swuWPcgZ4l9PGm61H3CFO4ov0Ski9NMKd09ve7BWBI6peVFZRQDiagySGmV7ao3XjSFB6U3GhcBnH03aNvQDKk749aS1H
tEd9W6Bbz2eCT3gFc73HfxKph58kkdq3NMp7+xtYhDO8MItt0++hzSlqBP0tlTkOydcQg=="}
Current votes: 1 (Registered)
```

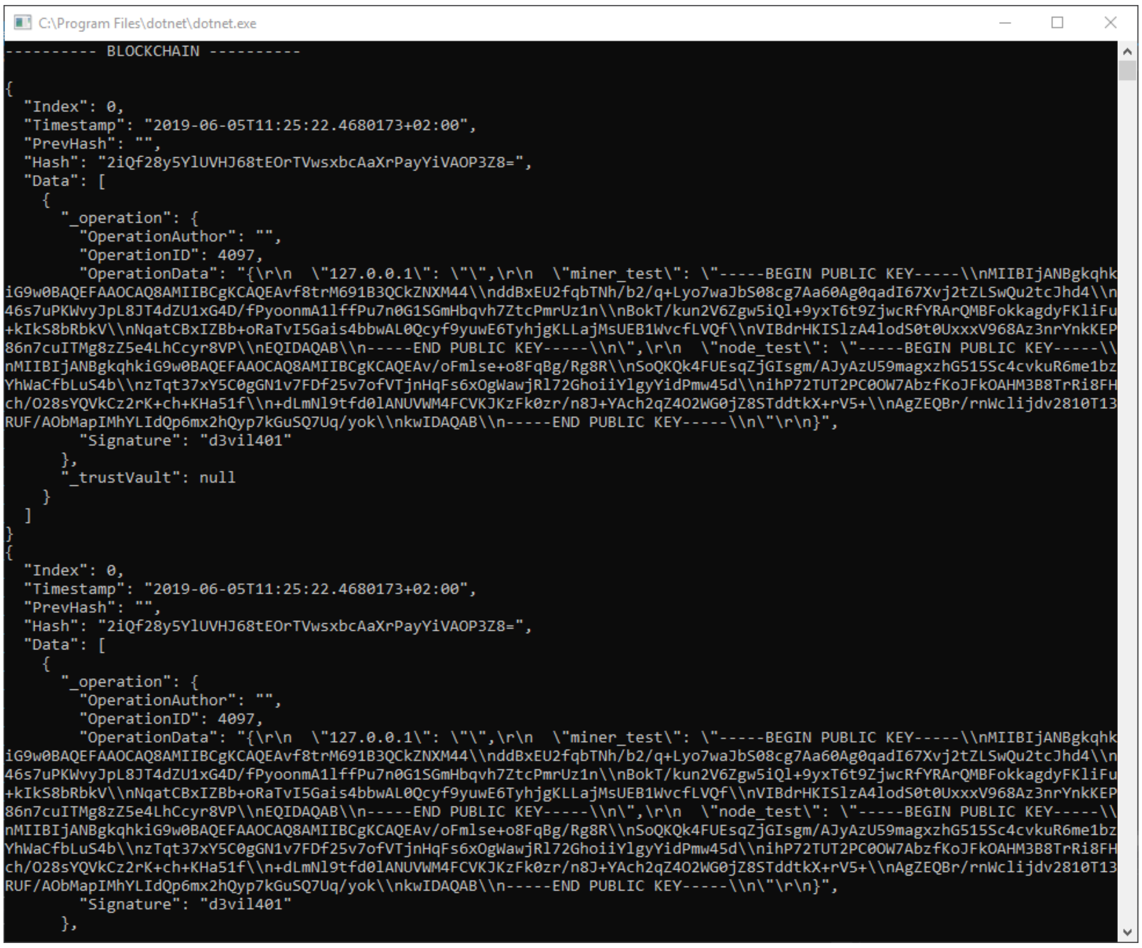
The miner connecting to the node, authenticating itself, placing a transaction request and requesting a pending operation.

The transaction is in fact dumped on the console ("Transaction by…") and the vote is casted.

After, the miner impersonates the administrator and prints the latest block information received from the node (statistics).

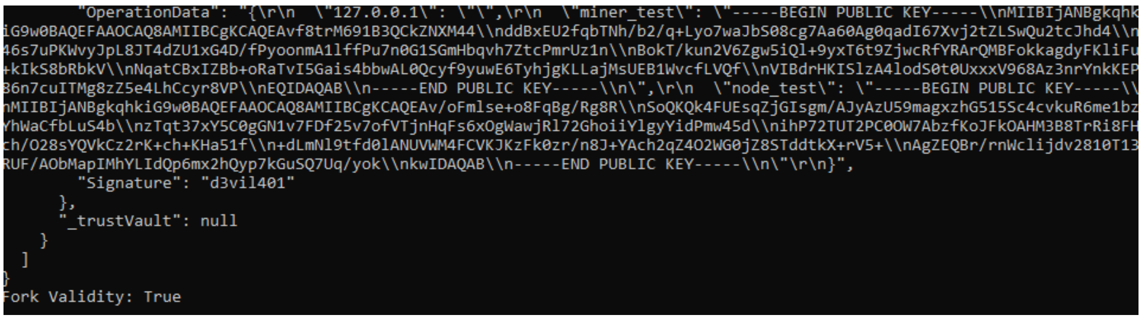
In the end, the miner uses the previously obtained TxID and queries the status of the transaction previously voted to check if it has been confirmed and registered (it can in fact be seen in the statistics' dump).

#### 5.1.1.2 Node Test


C:\Program Files\dotnet\dotnet.exe

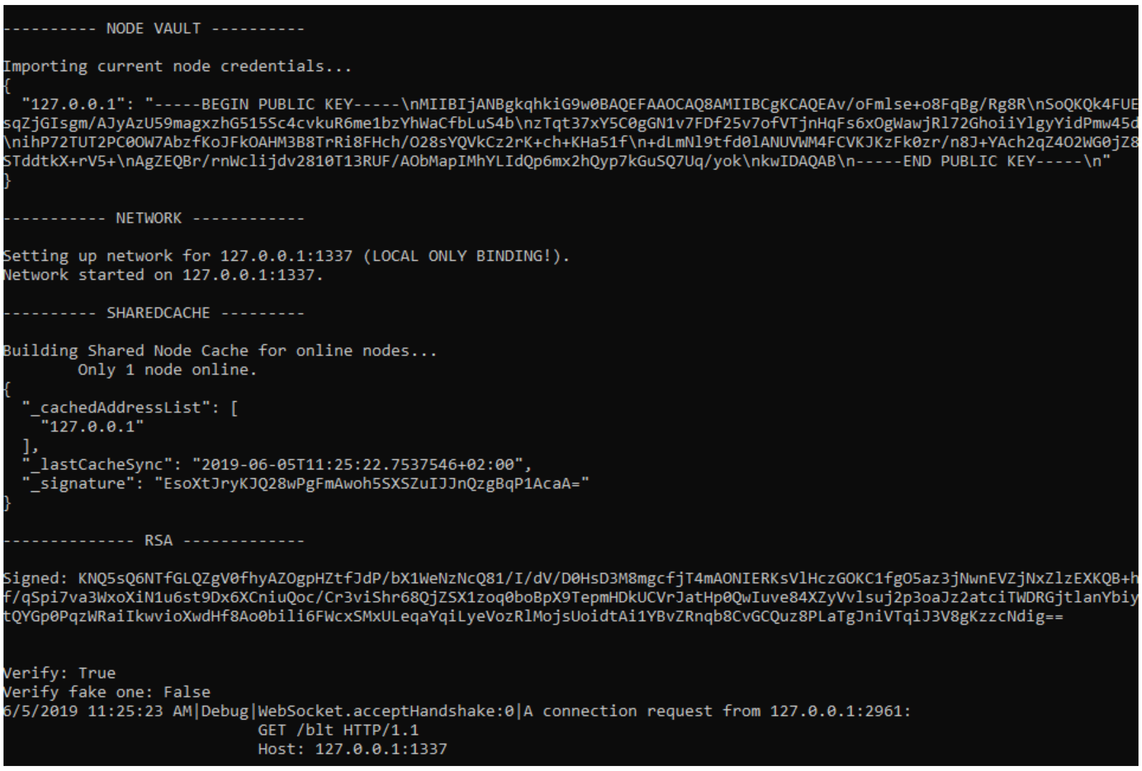
```
---------- BLOCKCHAIN ----------

{
  "Index": 0,
  "Timestamp": "2019-06-05T11:25:22.4680173+02:00",
  "PrevHash": "",
  "Hash": "2iQf28y5YlUVHJ68tEOrTVwsxbcAaXrPayYiVAOP3Z8=",
  "Data": [
    {
      "_operation": {
        "OperationAuthor": "",
        "OperationID": 4097,
        "OperationData": "{\r\n  \"127.0.0.1\": \"\",\r\n  \"miner_test\": \"-----BEGIN PUBLIC KEY-----\\nMIIBIjANBgkqhk
iG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAvf8trM691B3QCkZNXM44\\nddBxEU2fqbTNh/b2/q+Lyo7waJbS08cg7Aa60Ag0qadI67Xvj2tZLSwQu2tcJhd4\\n
46s7uPKWvyJpL8JT4dZU1xG4D/fPyoonmA1lffPu7n0G1SGmHbqvh7ZtcPmrUz1n\\nBokT/kun2V6Zgw5iQl+9yxT6t9ZjwcRfYRArQMBFokkagdyFKliFu
+kIkS8bRbkV\\nNqatCBxIZBb+oRaTvI5Gais4bbwAL0Qcyf9yuwE6TyhjgKLLajMsUEB1WvcfLVQf\\nVIBdrHKISlzA4lodS0t0UxxxV968Az3nrYnkKEP
86n7cuITMg8zZ5e4LhCcyr8VP\\nEQIDAQAB\\n-----END PUBLIC KEY-----\\n\",\r\n  \"node_test\": \"-----BEGIN PUBLIC KEY-----\\
nMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAv/oFmlse+o8FqBg/Rg8R\\nSoQKQk4FUEsqZjGIsgm/AJyAzU59magxzhG515Sc4cvkuR6me1bz
YhWaCfbLuS4b\\nzTqt37xY5C0gGN1v7FDf25v7ofVTjnHqFs6xOgWawjRl72GhoiiYlgyYidPmw45d\\nihP72TUT2PC0OW7AbzfKoJFkOAHM3B8TrRi8FH
ch/O28sYQVkCz2rK+ch+KHa51f\\n+dLmNl9tfd0lANUVWM4FCVKJKzFk0zr/n8J+YAch2qZ4O2WG0jZ8STddtkX+rV5+\\nAgZEQBr/rnWclijdv2810T13
RUF/AObMapIMhYLIdQp6mx2hQyp7kGuSQ7Uq/yok\\nkwIDAQAB\\n-----END PUBLIC KEY-----\\n\"\r\n}",
        "Signature": "d3vil401"
      },
      "_trustVault": null
    }
  ]
}
{
  "Index": 0,
  "Timestamp": "2019-06-05T11:25:22.4680173+02:00",
  "PrevHash": "",
  "Hash": "2iQf28y5YlUVHJ68tEOrTVwsxbcAaXrPayYiVAOP3Z8=",
  "Data": [
    {
      "_operation": {
        "OperationAuthor": "",
        "OperationID": 4097,
        "OperationData": "{\r\n  \"127.0.0.1\": \"\",\r\n  \"miner_test\": \"-----BEGIN PUBLIC KEY-----\\nMIIBIjANBgkqhk
iG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAvf8trM691B3QCkZNXM44\\nddBxEU2fqbTNh/b2/q+Lyo7waJbS08cg7Aa60Ag0qadI67Xvj2tZLSwQu2tcJhd4\\n
46s7uPKWvyJpL8JT4dZU1xG4D/fPyoonmA1lffPu7n0G1SGmHbqvh7ZtcPmrUz1n\\nBokT/kun2V6Zgw5iQl+9yxT6t9ZjwcRfYRArQMBFokkagdyFKliFu
+kIkS8bRbkV\\nNqatCBxIZBb+oRaTvI5Gais4bbwAL0Qcyf9yuwE6TyhjgKLLajMsUEB1WvcfLVQf\\nVIBdrHKISlzA4lodS0t0UxxxV968Az3nrYnkKEP
86n7cuITMg8zZ5e4LhCcyr8VP\\nEQIDAQAB\\n-----END PUBLIC KEY-----\\n\",\r\n  \"node_test\": \"-----BEGIN PUBLIC KEY-----\\
nMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAv/oFmlse+o8FqBg/Rg8R\\nSoQKQk4FUEsqZjGIsgm/AJyAzU59magxzhG515Sc4cvkuR6me1bz
YhWaCfbLuS4b\\nzTqt37xY5C0gGN1v7FDf25v7ofVTjnHqFs6xOgWawjRl72GhoiiYlgyYidPmw45d\\nihP72TUT2PC0OW7AbzfKoJFkOAHM3B8TrRi8FH
ch/O28sYQVkCz2rK+ch+KHa51f\\n+dLmNl9tfd0lANUVWM4FCVKJKzFk0zr/n8J+YAch2qZ4O2WG0jZ8STddtkX+rV5+\\nAgZEQBr/rnWclijdv2810T13
RUF/AObMapIMhYLIdQp6mx2hQyp7kGuSQ7Uq/yok\\nkwIDAQAB\\n-----END PUBLIC KEY-----\\n\"\r\n}",
        "Signature": "d3vil401"
      },
```

The Node test dumping the blockchain’s structure (genesis block) after set-up.



```
        "OperationData": "{\r\n  \"127.0.0.1\": \"\",\r\n  \"miner_test\": \"-----BEGIN PUBLIC KEY-----\\nMIIBIjANBgkqhk
iG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAvf8trM691B3QCkZNXM44\\nddBxEU2fqbTNh/b2/q+Lyo7waJbS08cg7Aa60Ag0qadI67Xvj2tZLSwQu2tcJhd4\\n
46s7uPKWvyJpL8JT4dZU1xG4D/fPyoonmA1lffPu7n0G1SGmHbqvh7ZtcPmrUz1n\\nBokT/kun2V6Zgw5iQl+9yxT6t9ZjwcRfYRArQMBFokkagdyFKliFu
+kIkS8bRbkV\\nNqatCBxIZBb+oRaTvI5Gais4bbwAL0Qcyf9yuwE6TyhjgKLLajMsUEB1WvcfLVQf\\nVIBdrHKISlzA4lodS0t0UxxxV968Az3nrYnkKEP
86n7cuITMg8zZ5e4LhCcyr8VP\\nEQIDAQAB\\n-----END PUBLIC KEY-----\\n\",\r\n  \"node_test\": \"-----BEGIN PUBLIC KEY-----\\
nMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAv/oFmlse+o8FqBg/Rg8R\\nSoQKQk4FUEsqZjGIsgm/AJyAzU59magxzhG515Sc4cvkuR6me1bz
YhWaCfbLuS4b\\nzTqt37xY5C0gGN1v7FDf25v7ofVTjnHqFs6xOgWawjRl72GhoiiYlgyYidPmw45d\\nihP72TUT2PC0OW7AbzfKoJFkOAHM3B8TrRi8FH
ch/O28sYQVkCz2rK+ch+KHa51f\\n+dLmNl9tfd0lANUVWM4FCVKJKzFk0zr/n8J+YAch2qZ4O2WG0jZ8STddtkX+rV5+\\nAgZEQBr/rnWclijdv2810T13
RUF/AObMapIMhYLIdQp6mx2hQyp7kGuSQ7Uq/yok\\nkwIDAQAB\\n-----END PUBLIC KEY-----\\n\"\r\n}",
        "Signature": "d3vil401"
      },
      "_trustVault": null
    }
  ]
}
Fork Validity: True
```

If the fork is valid, it marks a success as the structure of the blockchain is regular and as expected.

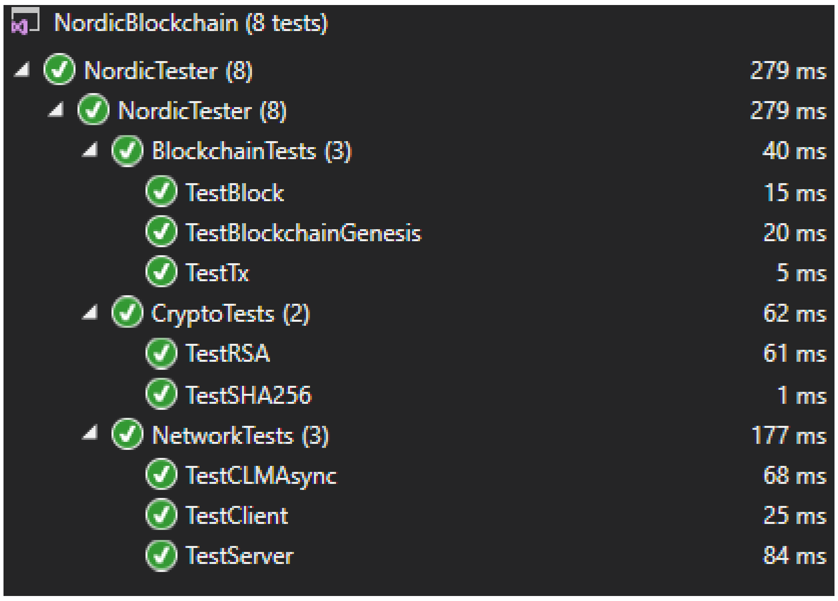


```
---------- NODE VAULT ----------

Importing current node credentials...
{
  "127.0.0.1": "-----BEGIN PUBLIC KEY-----\nMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAv/oFmlse+o8FqBg/Rg8R\nSoQKQk4FUE
sqZjGIsgm/AJyAzU59magxzhG515Sc4cvkuR6me1bzYhWaCfbLuS4b\nzTqt37xY5C0gGN1v7FDf25v7ofVTjnHqFs6xOgWawjRl72GhoiiYlgyYidPmw45d
\nihP72TUT2PC0OW7AbzfKoJFkOAHM3B8TrRi8FHch/O28sYQVkCz2rK+ch+KHa51f\n+dLmNl9tfd0lANUVWM4FCVKJKzFk0zr/n8J+YAch2qZ4O2WG0jZ8
STddtkX+rV5+\nAgZEQBr/rnWclijdv2810T13RUF/AObMapIMhYLIdQp6mx2hQyp7kGuSQ7Uq/yok\nkwIDAQAB\n-----END PUBLIC KEY-----\n"
}

----------- NETWORK -----------

Setting up network for 127.0.0.1:1337 (LOCAL ONLY BINDING!).
Network started on 127.0.0.1:1337.

---------- SHAREDCACHE ---------

Building Shared Node Cache for online nodes...
        Only 1 node online.
{
  "_cachedAddressList": [
    "127.0.0.1"
  ],
  "_lastCacheSync": "2019-06-05T11:25:22.7537546+02:00",
  "_signature": "EsoXtJryKJQ28wPgFmAwoh5SXSZuIJJnQzgBqP1AcaA="
}

-------------- RSA -------------

Signed: KNQ5sQ6NTfGLQZgV0fhyAZOgpHZtfJdP/bX1WeNzNcQ81/I/dV/D0HsD3M8mgcfjT4mAONIERKsVlHczGOKC1fgO5az3jNwnEVZjNxZlzEXKQB+h
f/qSpi7va3WxoXiN1u6st9Dx6XCniuQoc/Cr3viShr68QjZSX1zoq0boBpX9TepmHDkUCVrJatHp0QwIuve84XZyVvlsuj2p3oaJz2atciTWDRGjtlanYbiy
tQYGp0PqzWRaiIkwvioXwdHf8Ao0bili6FWcxSMxULeqaYqiLyeVozRlMojsUoidtAi1YBvZRnqb8CvGCQuz8PLaTgJniVTqiJ3V8gKzzcNdig==


Verify: True
Verify fake one: False
6/5/2019 11:25:23 AM|Debug|WebSocket.acceptHandshake:0|A connection request from 127.0.0.1:2961:
                           GET /blt HTTP/1.1
                           Host: 127.0.0.1:1337
```

The remaining components of the node being checked before starting the server.

If the server started and responding to the miner’s attempts of connection, the node is fully functional.

#### 5.1.1.3 Unit-Tests



```
NordicBlockchain (8 tests)
◢ ✓ NordicTester (8)                          279 ms
  ◢ ✓ NordicTester (8)                        279 ms
    ◢ ✓ BlockchainTests (3)                    40 ms
        ✓ TestBlock                            15 ms
        ✓ TestBlockchainGenesis                20 ms
        ✓ TestTx                                5 ms
    ◢ ✓ CryptoTests (2)                        62 ms
        ✓ TestRSA                              61 ms
        ✓ TestSHA256                            1 ms
    ◢ ✓ NetworkTests (3)                      177 ms
        ✓ TestCLMAsync                         68 ms
        ✓ TestClient                           25 ms
        ✓ TestServer                           84 ms
```

The unit-tests are used to test the single components regardless of their role in the whole project, the description of these tests is below.

### 5.1.2 Testing practices

Since the project is mostly back-end type, the White Box testing methodology has been chosen, by implementing and using the results of Unit Testing for the single components; while for the node and miner's (including requirements) tests an ad-hoc application for manual debugging has been created.
For Unit Testing, the X-Unit testing framework was the only compatible one with .NET Core that was also by default included in the installation of the .NET subsystem.
A brief description of the unit tests developed:

- Network functionalities:
  - Test server creation.
  - Test client creation and connection with the server, using an IOperation for transmission test.
  - Test CLM Manager, by creating a request and expecting a correct response.
- Blockchain functionalities:

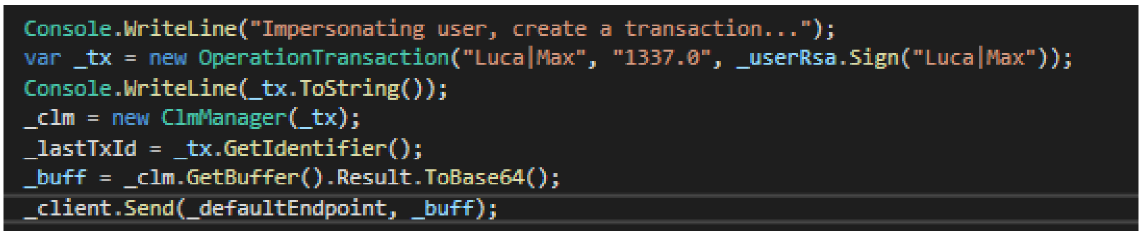
  - Test blockchain creation and genesis block.
  - Test transaction requests, vote casting, confirming status and status request.
  - Test block creation, block registration, block querying and blockchain compromise.
- Cryptographic functionalities:
  - Test RSA private and public key importation, signing and signature verification.
  - Test SHA-256 creation, hash calculation and hash comparison including expected hash result comparison (hardcoded).

Testing the functional requirements required a different approach than Unit Testing and an ad-hoc project has been created for such tasks (“NordicMiner” and “NBService”), containing the required code using the functional requirement.
These should be verified trough debugging due to the structure of the dedicated classes (CLM Manager) since the exceptions thrown are designed not to expose too many information of traveling data and due to the complexity of the flow of code involved into the various different classes.

## 5.2 Test Specifications

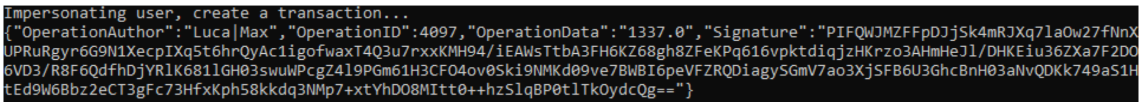
### 5.2.1 Specification 1: User’s transaction placement, confirmation and verification.

The test should emulate the user’s placement of an OperationTransaction in the blockchain (Figure 4).
If the transaction has been put into the pending operations, cast a vote and expect it to be confirmed definitively and verify that it has been placed in the latest block.
This is tested into the “Nordic Miner” project.



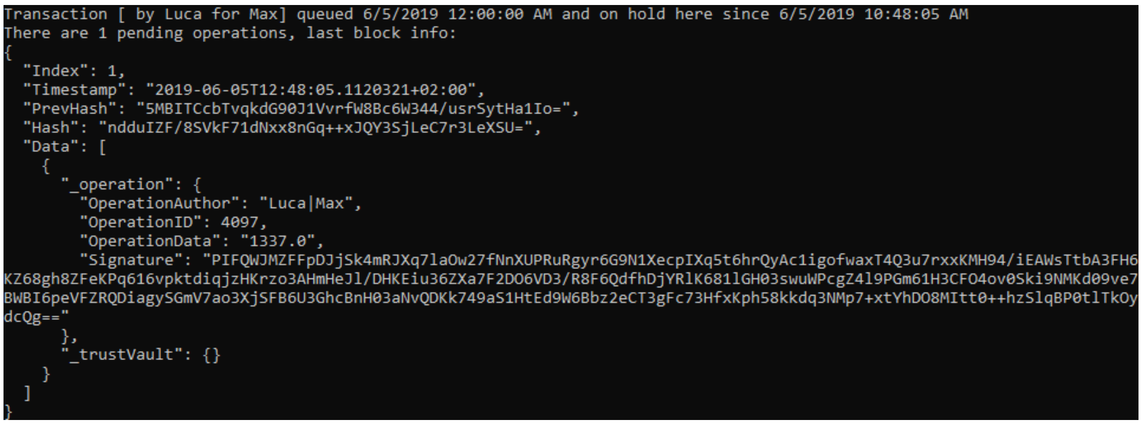
```
Console.WriteLine("Impersonating user, create a transaction...");
var _tx = new OperationTransaction("Luca|Max", "1337.0", _userRsa.Sign("Luca|Max"));
Console.WriteLine(_tx.ToString());
_clm = new ClmManager(_tx);
_lastTxId = _tx.GetIdentifier();
_buff = _clm.GetBuffer().Result.ToBase64();
_client.Send(_defaultEndpoint, _buff);
```

*Figure 4 - The code creating a transaction request*



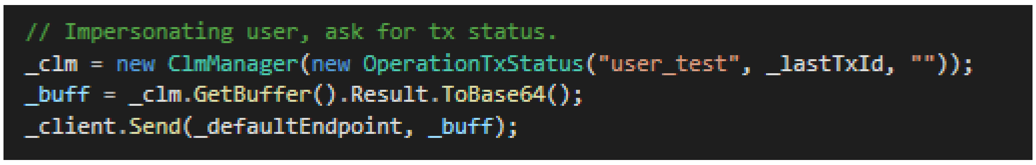
```
Impersonating user, create a transaction...
{"OperationAuthor":"Luca|Max","OperationID":4097,"OperationData":"1337.0","Signature":"PIFQWJMZFFpDJjSk4mRJXq7laOw27fNnX
UPRuRgyr6G9N1XecpIXq5t6hrQyAc1igofwaxT4Q3u7rxxKMH94/iEAWsTtbA3FH6KZ68gh8ZFeKPq616vpktdiqjzHKrzo3AHmHeJl/DHKEiu36ZXa7F2DO
6VD3/R8F6QdfhDjYRlK681lGH03swuWPcgZ4l9PGm61H3CFO4ov0Ski9NMKd09ve7BWBI6peVFZRQDiagySGmV7ao3XjSFB6U3GhcBnH03aNvQDKk749aS1H
tEd9W6Bbz2eCT3gFc73HfxKph58kkdq3NMp7+xtYhDO8MItt0++hzSlqBP0tlTkOydcQg=="}
```

*Figure 5 - The resulting output in the console at the moment of creation*



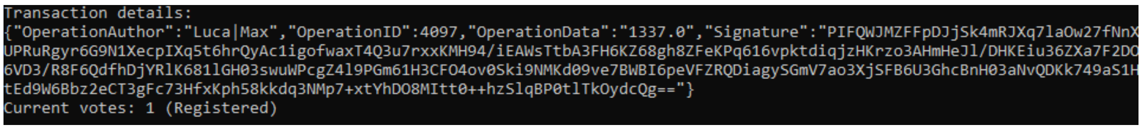
```
Transaction [ by Luca for Max] queued 6/5/2019 12:00:00 AM and on hold here since 6/5/2019 10:48:05 AM
There are 1 pending operations, last block info:
{
  "Index": 1,
  "Timestamp": "2019-06-05T12:48:05.1120321+02:00",
  "PrevHash": "5MBITCcbTvqkdG90J1VvrfW8Bc6W344/usrSytHa1Io=",
  "Hash": "ndduIZF/8SVkF71dNxx8nGq++xJQY3SjLeC7r3LeXSU=",
  "Data": [
    {
      "_operation": {
        "OperationAuthor": "Luca|Max",
        "OperationID": 4097,
        "OperationData": "1337.0",
        "Signature": "PIFQWJMZFFpDJjSk4mRJXq7laOw27fNnXUPRuRgyr6G9N1XecpIXq5t6hrQyAc1igofwaxT4Q3u7rxxKMH94/iEAWsTtbA3FH6
KZ68gh8ZFeKPq616vpktdiqjzHKrzo3AHmHeJl/DHKEiu36ZXa7F2DO6VD3/R8F6QdfhDjYRlK681lGH03swuWPcgZ4l9PGm61H3CFO4ov0Ski9NMKd09ve7
BWBI6peVFZRQDiagySGmV7ao3XjSFB6U3GhcBnH03aNvQDKk749aS1HtEd9W6Bbz2eCT3gFc73HfxKph58kkdq3NMp7+xtYhDO8MItt0++hzSlqBP0tlTkOy
dcQg=="
      },
      "_trustVault": {}
    }
  ]
}
```

*Figure 6 – The confirmation that the node received the transaction request that has been already processed, using the administration's statistics request to dump the block's structure.*



```
// Impersonating user, ask for tx status.
_clm = new ClmManager(new OperationTxStatus("user_test", _lastTxId, ""));
_buff = _clm.GetBuffer().Result.ToBase64();
_client.Send(_defaultEndpoint, _buff);
```

*Figure 7 - The code for the user's request of status of the previous transaction*

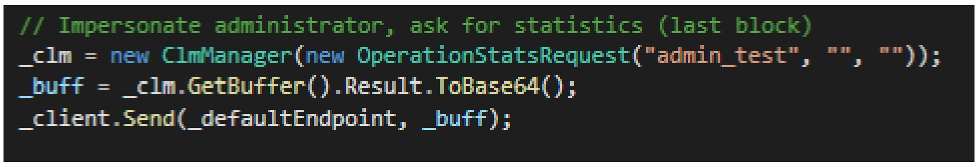


```
Transaction details:
{"OperationAuthor":"Luca|Max","OperationID":4097,"OperationData":"1337.0","Signature":"PIFQWJMZFFpDJjSk4mRJXq7laOw27fNnX
UPRuRgyr6G9N1XecpIXq5t6hrQyAc1igofwaxT4Q3u7rxxKMH94/iEAWsTtbA3FH6KZ68gh8ZFeKPq616vpktdiqjzHKrzo3AHmHeJl/DHKEiu36ZXa7F2DO
6VD3/R8F6QdfhDjYRlK681lGH03swuWPcgZ4l9PGm61H3CFO4ov0Ski9NMKd09ve7BWBI6peVFZRQDiagySGmV7ao3XjSFB6U3GhcBnH03aNvQDKk749aS1H
tEd9W6Bbz2eCT3gFc73HfxKph58kkdq3NMp7+xtYhDO8MItt0++hzSlqBP0tlTkOydcQg=="}
Current votes: 1 (Registered)
```

*Figure 8 - The transaction status response from the previous user's request*

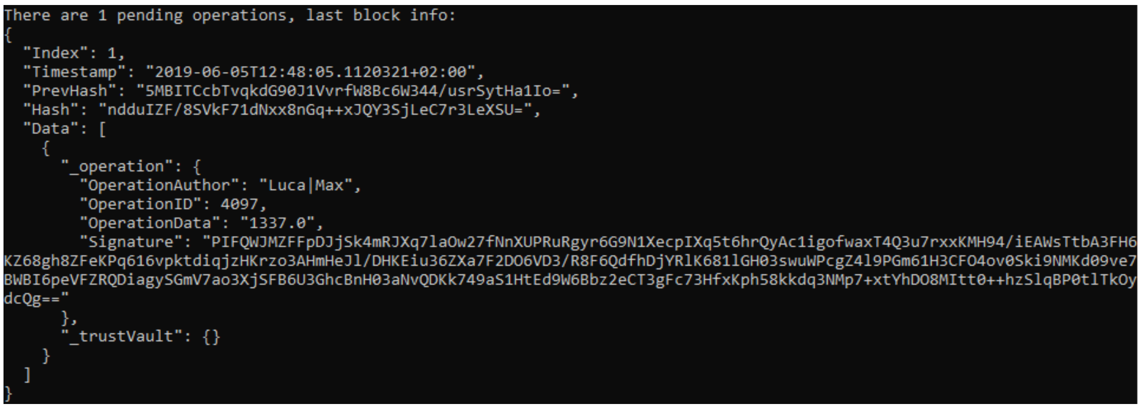
### 5.2.2 Specification 2: Administrator's request of statistics.

The test must use the Client class, after a server instance has been created, to send a request for statistics and expect a response with a compressed string corresponding to the JSON representation of the blockchain's last block and a number representing the pending operations stored in the node.



```
// Impersonate administrator, ask for statistics (last block)
_clm = new ClmManager(new OperationStatsRequest("admin_test", "", ""));
_buff = _clm.GetBuffer().Result.ToBase64();
_client.Send(_defaultEndpoint, _buff);
```

*Figure 9 - The code for the administrator's request of statistics*

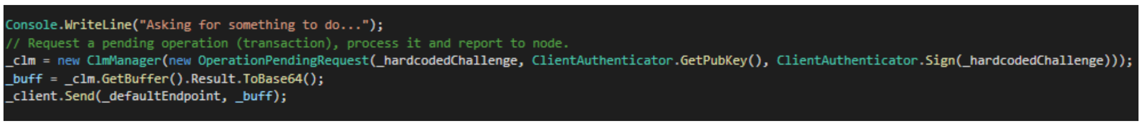


```
There are 1 pending operations, last block info:
{
  "Index": 1,
  "Timestamp": "2019-06-05T12:48:05.1120321+02:00",
  "PrevHash": "5MBITCcbTvqkdG90J1VvrfW8Bc6W344/usrSytHa1Io=",
  "Hash": "ndduIZF/8SVkF71dNxx8nGq++xJQY3SjLeC7r3LeXSU=",
  "Data": [
    {
      "_operation": {
        "OperationAuthor": "Luca|Max",
        "OperationID": 4097,
        "OperationData": "1337.0",
        "Signature": "PIFQWJMZFFpDJjSk4mRJXq7laOw27fNnXUPRuRgyr6G9N1XecpIXq5t6hrQyAc1igofwaxT4Q3u7rxxKMH94/iEAWsTtbA3FH6
KZ68gh8ZFeKPq616vpktdiqjzHKrzo3AHmHeJl/DHKEiu36ZXa7F2DO6VD3/R8F6QdfhDjYRlK681lGH03swuWPcgZ4l9PGm61H3CFO4ov0Ski9NMKd09ve7
BWBI6peVFZRQDiagySGmV7ao3XjSFB6U3GhcBnH03aNvQDKk749aS1HtEd9W6Bbz2eCT3gFc73HfxKph58kkdq3NMp7+xtYhDO8MItt0++hzSlqBP0tlTkOy
dcQg=="
      },
      "_trustVault": {}
    }
  ]
}
```

*Figure 10 - The console output of the statistics request, after a transaction has been already registered*

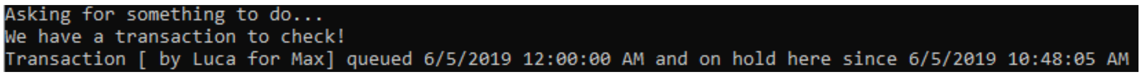
### 5.2.3 Specification 3: Miner's fetch of pending operation (transaction) and confirmation.

The test must use the Client class, after a server has been created and a transaction request has been placed, to request a pending operation to be processed and verify that the pending operation received is the same as the one registered (by identifier comparison).
Once the operation is received, the miner must cast a vote and a transaction status request must be sent to the server to verify that the operation has been confirmed and placed in a block.



```
Console.WriteLine("Asking for something to do...");
// Request a pending operation (transaction), process it and report to node.
_clm = new ClmManager(new OperationPendingRequest(_hardcodedChallenge, ClientAuthenticator.GetPubKey(), ClientAuthenticator.Sign(_hardcodedChallenge)));
_buff = _clm.GetBuffer().Result.ToBase64();
_client.Send(_defaultEndpoint, _buff);
```

*Figure 11 - The code for the miner's fetching of a pending operation*



```
Asking for something to do...
We have a transaction to check!
Transaction [ by Luca for Max] queued 6/5/2019 12:00:00 AM and on hold here since 6/5/2019 10:48:05 AM
```

*Figure 12 - The output on the console of the retrieved pending operation (a transaction previously submitted)*

# 6 Results and Discussion

The development process for Nordic Blockchain concluded into a creation of a functional proof of concept prototype that satisfies the functional requirements, validating its goal. The blockchain structure has proven to be ready for usage in a production quality environment as it is not a complex structure and has been properly tested.
Certain operations, such as the cryptographic ones, are not reliable due to the lack of time and resources allocated to the project development but are not necessary to justify the existence of the project goal itself.
Certain aspects of the project’s structure and design, during development, had to be hardcoded or simplified due to the lack of funding, personnel and time till the deadline.
A few discussion points are listed below:

- The cryptography class “RSA” is working if taken singularly, but do not work properly when adopted into the whole mechanism, the reason is still unclear.

- The “TrustVault” is working as intended and is adopted where possible but lacks an importation method from a file to allow persistence of public keys database.

- The miner and node require better ad-hoc testing platform that rely on testing frameworks and the miner should not test the administration and user’s operations.

- The miner’s purpose, on the contrary of traditional blockchain systems, is not to keep decentralization focus but to only automate the process of transactions verification; decentralization, in a context of full-control and monitorization, is weak.
  Despite the limitations, the Node role changes from point of storage to authoritative broadcasting of operations.

- A strength point for eventual rogue miners is to target and confirm specific malicious transactions; the design approached here does not in fact solve or limit the 51% attack but the usage of asymmetric keys allows the nodes to only recognize and trust miners that have been previously put into the trust list (ClientAuthenticator).

- The CLM Manager is a very specific implementation-oriented class and requires lots of care when implementing new operation types; a better and more generic implementation can be achieved if slightly reworked.

- The SessionHandler class is not properly implemented, no unique identifier is associated with existing sessions and no server-side session control exists due to limitations of used APIs; development of custom WebSockets API will unlock another area of server-controls.

- Persistence of blockchain hasn’t been planned in this project, but the usage of JSON will speed up future implementations once implemented the importing methods.

- The signing and verification trough RSA should be done over the SHA256 hash of the content, not on the content directly.

- The nodes should communicate with each other and be relay point for the broadcast messages, a base implementation for the broadcasts exist but hasn't been adopted nor approached in the current code flow.

# 7 Conclusions

In conclusion, Nordic Blockchain's development was satisfactory, despite the lack of manpower. The blockchain is proven to be a scalable and secure storage system and easy to backup, provided the stakeholder will implement its persistence.

The author had different additional ideas over the networking and synchronization functionalities that had to be sacrificed, but traces of implementation are left in the code for future development.

Further research should be made into the security side of the project and the front-end part of the project for user's interaction.

A test-driven development should be employed rigorously due to criticality of safety nature of the project itself.

The system's structure offered interesting results for a baseline of development for future implementation of verify-and-record mechanisms that are very common in government and administration areas.

Overall, the project is considered a success that offered and completed the promises and goals set, despite its weaknesses in certain areas.

# 8 Project Future

Nordic Blockchain is just the first approach with the mindset of a prototype, to evolve such project furtherly, certain aspects may need a rework while others need to be developed with a certain priority:

- The cryptographic classes and their adoption should be reworked completely.

- The IOperation class should be reworked slightly to include better “OperationTransaction” definition of fields (sender, receiver, …) instead of using joined strings.
- A fundamental safer mechanism for transaction would be the usage of Smart Contracts, that would bring even a more modern approach to the original problem.
- The blockchain structure is a solid design but may be improved over the time by adding additional dependency for validity such as double-sided hash storage (store both the previous block’s hash and the next block’s hash), albeit enforcing the Block security trough additional dependency factors.
- The miner should be authenticated every time.
- The broadcasting mechanism and inter-node communication should be created.
- The synchronization between nodes should be created, on top of the broadcasting mechanism; this is fundamental for the flexibility of the blockchain towards offline nodes.
- The implementation of a wallet system will allow user identification and perhaps the registration of assets directly on the blockchain.

The project may not be reused for direct development but may offer a strong basis for a future implementation of a blockchain in banking systems, given the right resources and personnel.

Perhaps, stakeholders can extend the interest area of this project baseline to a multiplicity of implementation possibilities and experiments, offering limitless possibilities and provoking discussion or ideas for different purposes.

# 9 Appendices

- BlockchainUC.asta – Use Cases, Use case descriptions.
- Overall.png – Overall class diagram, full picture.
- ClassDiagrams.asta – Class diagrams, Sequence Diagrams.

- NordicBlockchain.zip – Project implementation (code, binaries, misc).

# 10 Sources of information

Henry He, 2018. *Building A Blockchain In .NET Core*. Available at: https://www.c-sharpcorner.com/article/building-a-blockchain-in-net-core-proof-of-work/ [Accessed on 6 June 2019]

Ana Berman, 2018. *Private Blockchains Could Be Compatible with EU Privacy Rules, Research Shows*. Available at: https://cointelegraph.com/news/private-blockchains-could-be-compatible-with-eu-privacy-rules-research-shows [Accessed on 6 June 2019]

Jake Frankenfield, 2019. *51% Attack*. Available at: https://www.investopedia.com/terms/1/51-attack.asp [Accessed on 6 June 2019]

Ivan Ristic, 2012. *CRIME: Information Leakage Attack against SSL/TLS*. Available at: https://blog.qualys.com/ssllabs/2012/09/14/crime-information-leakage-attack-against-ssltls [Accessed on 6 June 2019]

Software Testing Fundamentals, 2019*. Unit Testing*. Available at: http://softwaretestingfundamentals.com/unit-testing/ [Accessed on 6 June 2019]

Stack Overflow, 2019. *Most Popular Technologies*. Available at: https://insights.stackoverflow.com/survey/2019#technology [Accessed 28 May, 2019].

GeeksforGeeks, 2019. *RSA Algorithm in Cryptography*. Available at: https://www.geeksforgeeks.org/rsa-algorithm-cryptography/ [Accessed on 6 June 2019]

Wikipedia, 2019. *Secure Hash Algorithms*. Available at: https://en.wikipedia.org/wiki/Secure_Hash_Algorithms [Accessed on 6 June 2019]

Accenture, 2019*. The future of blockchain banking*. Available at: https://www.accenture.com/us-en/insight-blockchain-technology-how-banks-building-real-time [Accessed on 6 June 2019]

IETF, 2011. *RFC 6455: The WebSocket Protocol*. Available at: https://tools.ietf.org/html/rfc6455 [Accessed on 6 June 2019]

Wikipedia, 2019. *Transport Layer Security*. Available at: https://it.wikipedia.org/wiki/Transport_Layer_Security [Accessed on 6 June 2019]

Ameer Rosic, 2016. *Smart Contracts: The Blockchain Technology That Will Replace Lawyers*. Available at: https://blockgeeks.com/guides/smart-contracts/ [Accessed on 6 June 2019]

Wikipedia, 2019. JavaScript Object Notation. Available at: https://it.wikipedia.org/wiki/JavaScript_Object_Notation [Accessed on 6 June 2019]

IEEE Computer Society, 2008. *IEEE Std 829-2008, IEEE Standard for Software and System Test Documentation*.

Larman, C., 2004. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development*.

Wikipedia, 2019. *Kanabn (development)*. Available at: https://en.wikipedia.org/wiki/Kanban_(development) [Accessed on 6 June 2019]