Bachelorarbeit

BA05 - Implement private blockchain system

Verfasser

Dejan Micic

angestrebter akademischer Grad

Bachelor of Science (BSc)

Wien, 2023

Studienkennzahl lt. Studienblatt:	A 526 [3]
Fachrichtung:	Informatik - Wirtschaftsinformatik
Betreuerin / Betreuer:	Dr.techn. Belal Abu Naim

Contents

Abstract

Blockchain is a widely adopted technology that has garnered significant attention across various industries. Within the area of security measures, it can be categorized into public, private, consortium and hybrid blockchains. The objective of this project is to provide a full understanding of blockchain technology by implementing a small-scale private blockchain project. This project specifically focuses on a use-case scenario involving a library, where users/peers have the capability to contribute books. Each book entry includes relevant information such as the book's author, title, publication date, and ISBN. Peers interact with the blockchain system through a command-line interface (CLI), which offers a selection of options to manipulate with the blockchain. Upon registering their ports, every individual peer gains the ability to add books to a shared pool. However, if a book with a particular ISBN has already been added, it is rejected. Once four books have been accumulated, one peer is randomly selected as the "winning" peer, responsible for creating and mining the block on the blockchain. All peers have access to the blockchain and can view all previously added blocks. This grants peers the ability to obtain a comprehensive overview of all books associated with a specific author through a search function. In conclusion, this thesis provides a brief explanation of the functioning of a simple private blockchain. It explores the core principles and mechanisms, offering clear insights into their operational processes. This project will bring light into understanding the secure and useful usability of blockchain, which will play an important role in the future.

1 Introduction

Since blockchain appeared on the market, the technology started innovating and opening new potential to growing various industries. The blockchain is mostly connected with cryptocurrencies like bitcoin, ethereum and others, the private blockchain offers better benefits that are good security and different to kind of approach on how the blockchain is functioning. Blockchains are a type of Distributed Ledger Technology(DLT) whose transactions are all digital and decentralized. The data is only transferred between the participants of the network. [14]

The main difference between private and public blockchain is that in private blockchain, the peers are the ones that control the restricted network. Only trusted users can manipulate with the blockchain, edit, add data to it[14]. There are three different kinds of blockchain:

- Public blockchain

- Private blockchain

- Consortium blockchain

Blockchain is normally built from multiple nodes, that don't trust each other. Sometimes there are nodes that have malicious intent but mostly all nodes are there for honest work. All nodes can perform transactions and modifications of the blockchain. Every node has the same order stored on the blockchain. They also agree on what can land on the blockchain.[4]

In case of Bitcoin, it uses the public blockchain as its implementation. Ethereum on the other hand uses the same consensus as bitcoin but with different parameters. Most of the implementations use the Proof of Work(PoW) as its protocol as where the private uses Proof of Authority(PoA) as its protocol. Besides those two there is also Proof of Stake(PoS). [4] [14]

The important difference between private and public blockchain is that the private blockchain is centralized. Because of that, private blockchain is good in use in various industries such as finance, supply chain management, healthcare, government and other industries that deal with very important information that is to be kept safe and secure.

Positive sides of private blockchains are:[8]

- All peers in the network agree on a consensus mechanism over which they send transaction, there is no middleman but the transaction is sent direct between the peers in the network.

- Transactions that are saved on the blockchain are all transparent and immutable, they cant be changed or manipulated. That is an important part for use-cases such as finance industries and other, which data has to be secure.

- All data that is saved in the blockchain has the same order between all peers on the network.

- Data on the blockchain is on the most side secured from manipulation and fraud. Therefore a good option for many industries.

1.1 Related work

Implementations that use similar type of use-case as used in this thesis are:

- Alexandria - Alexandria is a decentralized library which uses blockchain technology. Users are uploading and managing their work on the platform. [1]

- Publica.com - Peer to peer publishing network for authors. Publishers get initial coins on publishing a book, and readers can buy a digital copie of the book. It also ensures the immutability of the published books. [3]

- Po.et - users can timestamp their creations and publications on the blockchain.

With that they assure that they are legitimate owners of the work.[2]

2 State of the art

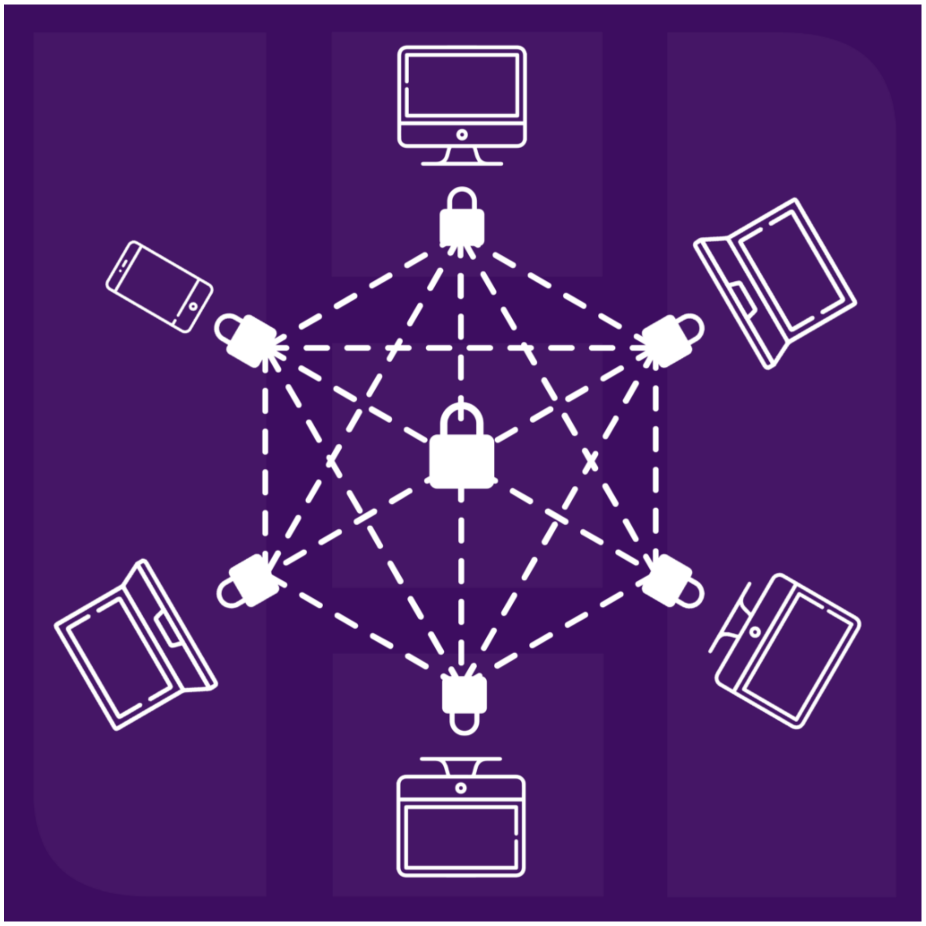


Figure 1: Private blockchain[10]

2.1 Consensus protocols

The consensus protocol plays one of the key roles that determine the kind of blockchain that is being made. In this layer of the blockchain, peers agree on what content is being appended to the blockchain. [4] When a peer wants to commit a block to the blockchain, all other peers that are connected to the network also have to make a copy of it in their blockchain database. As said before, there are three different kinds of consensus protocols. Those are:

- Proof of Work (PoW)

- Proof of Authority (PoA)

- Proof of Stake (PoS)

2.1.1 Proof of Work (PoW)

For block to be generated, a preimage of a hash number has to be calculated. It is used by most digital cryptocurrencies. All peers vote with its computing power and have to find a nonce value so that, when new blocks are added, the current hash has to be smaller then the target value.[5] On generating such value, the miners create the block and send it to the peers connected. The most important part of the Proof of Work consensus security is, that it relies on the fact, that no miner should compute more than 50% of the power. If it computes more, malicious intents could occur. The miner can then get more of the mining share and get the benefits of the mining process. [5]

2.1.2 Proof of Authority (PoA)

In this consensus protocol, there are peers that are set as authorities, and they are the only ones that can generate blocks. The most important key in Proof of authority is that the peers, that are considered as authority[4] are also considered as trustful and therefor it is mostly used for private blockchains. Industries such as Finance and governments with highly sensitive data mostly tend to use this kind of protocol because of it's safety and security.

2.1.3 Proof of Stake (PoS)

In the protocol Proof of Stake, the amount of share and voting power of the peers is established on the fact how much stake single peers own in the blockchain [13]. The positive side of this consensus protocol is, that the peers that computed most and contributed most to the blockchain have the most power on determining, if the blocks are valid. They are considered trustful because of the work they have already done.

2.2 Kinds of Blockchain

There are three different kinds of Blockchain.
We can differentiate between:

- Public blockchain

- Private blockchain

- Consortium blockchain

2.2.1 Public blockchain

Public blockchains are open for anyone. Anyone can join the network and validate transactions that are being made. Besides validating peers can also create transactions that are also public to anyone. Public blockchains use Proof of Work(PoW) as its consensus protocol. One of prime examples of public blockchain is Bitcoin.

2.2.2 Private blockchain

Only authorized peers can register to the network. There are strict rules on which peers can validate transactions which are being performed. Also only authorized users can perform the transactions. Private blockchain is using Proof of Authority(PoA) as its consensus protocol. It is mostly used by businesses, organisations and industries that require high security.

2.2.3 Consortium blockchain

The consortium blockchain is like private blockchain but more organisations have access to the data on the network. It is hybrid kind of blockchain that combines private and public. More users have access to the data but it is still mostly secure. It also uses Proof of Authority as its consensus protocol like private blockchain.

2.3 Centralized vs Decentralized

Decentralized blockchains mean that all peers work for themselves and there is no central organisation or group that controls everything in the network. Public blockchain uses decentralization because every single peer that connects can validate and create transaction on the blockchain. That leads to bad scalability. More peers that are connected to the network can lead to bad transaction speeds because of the time it needs to be validated and transferred to every single peer on the network.[9]

In the other hand, centralized blockchain, peers that have authority can only control what is happening on the blockchain. This method is better for blockchains that are going to save sensitive data. Good thing of decentralization is that the data is secure because only trusted peers are able to validate and create transactions.

2.4 Scalability and security

The importance of scalability of the blockchain plays an important role because it shows how good the blockchain can handle large amounts of transactions on the network on the same time. [11] If the scalability is bad, users can experience bottleneck and problems with the speed of the blockchain. This has a negative impact on the user experience and could hinder the blockchain from adapting and growing.

The scalability is calculated with the amount of transactions that are processed by the network every second [11]. With that information, developers can know, if the network has to be improved in favour of getting better scalability.

The security of the blockchain differs between private and public. Public networks normally allow anyone to access the network anonymously.[8] Those connected peers are then the ones that are validating the transactions that are created with its computing power. In the case of the public blockchain, the security is not the main part, because anyone can connect to the blockchain and validate the blocks. Because of the anonymity, peer can have malicious intents.

On the other hand, private blockchain, peers have identity and certain access privileges[8]. Only trusted peers that are considered authorised, have the ability to check the validity of the transactions.

3 Conceptual design

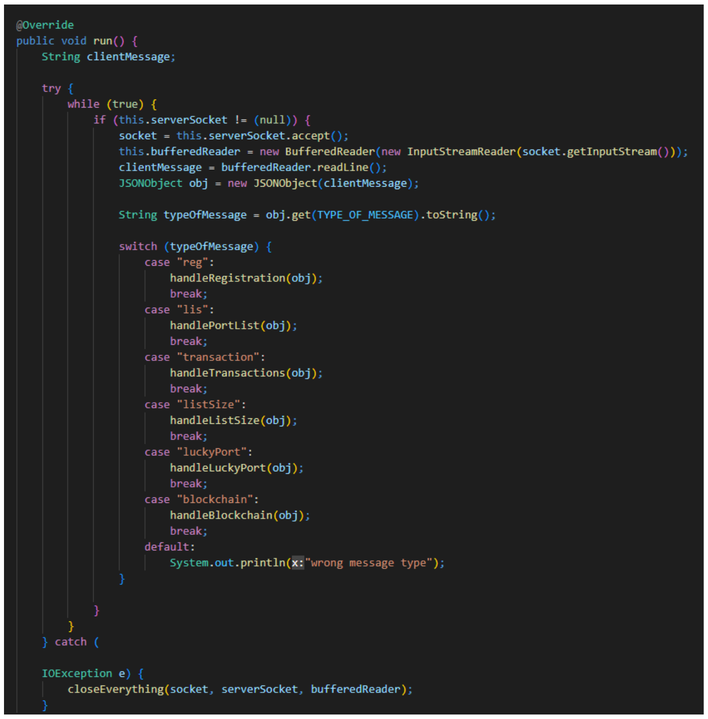
In this project, the private blockchain is being made for a library in which old important books are being kept. Peers that register to the blockchain network can add books my author, book name, publishing date and the ISBN of the book. After adding the book is forever kept on the blockchain with the information which peer added the book in the first place. Like this, authors register themselves as the owners of the book and register the book as available. The main idea is to keep track of what book is available at what peer. Another thing is that the blockchain can be searched for an author. Like this, users can find what books an author has in the library and can see which peer added it.
The first problem which has to be cleared is how to test any of the implementation. The idea here is to create the communication first whit which peers can communicate in the process of programming the project. Therefor the Peer to peer connection is being made firs. After that it has to be cleared what message type has to be transferred between peers. What technology should be used for the P2P communication. In this case Sockets are being used.
Next step is to see what messages are being transferred on the network for all the different steps from registration to transferring blocks on the blockchain.
Another point was to see if there would be an external database like SQL or SQL light or any other database, which was not implemented in this project, but linked lists are being used to store all necessary data.
It also should have been determined what data should be saved for books and what the blockchain should do for the peers, to be fully used. In that case, there can't be two or more Books with the same ISBN number. That assures that the Books are unique on the network.

3.1 Peer to peer network (P2P)

First of all, the network communication had to be implemented. Without the network, no communication could have been done, and therefor no further development would be possible. Blockchains are using peers, and they are communicating between each other and no server is needed. That is why for this implementation Peer to peer communication is most fitting. Every peer has its part for sending messages and the part for receiving messages. Like this, peers can send messages directly to other peers without being forced to send the messages to a server. Having the server as a middleman lowers the security of the communication between the peers that are registered to the network.

3.1.1 Asynchronous communication

To constantly be available and able to send and receive messages every peer has to work asynchronously. To get peers to work asynchronous, the TransactionServer is using threads. Threads give the peer the ability to receive the messages instantaneously without being blocked.



```java
@Override
public void run() {
    String clientMessage;

    try {
        while (true) {
            if (this.serverSocket != (null)) {
                socket = this.serverSocket.accept();
                this.bufferedReader = new BufferedReader(new InputStreamReader(socket.getInputStream()));
                clientMessage = bufferedReader.readLine();
                JSONObject obj = new JSONObject(clientMessage);

                String typeOfMessage = obj.get(TYPE_OF_MESSAGE).toString();

                switch (typeOfMessage) {
                    case "reg":
                        handleRegistration(obj);
                        break;
                    case "lis":
                        handlePortList(obj);
                        break;
                    case "transaction":
                        handleTransactions(obj);
                        break;
                    case "listSize":
                        handleListSize(obj);
                        break;
                    case "luckyPort":
                        handleLuckyPort(obj);
                        break;
                    case "blockchain":
                        handleBlockchain(obj);
                        break;
                    default:
                        System.out.println(x:"wrong message type");
                }

            }
        }
    } catch (

    IOException e) {
        closeEverything(socket, serverSocket, bufferedReader);
    }
```

Figure 2: Thread

3.1.2 Socket communication

To transfer messages between peers this implementation uses sockets. A sample of sending messages over sockets:
Private BufferedWriter bufferedWriter;
bufferedWriter.write(transactionObject.toString());
bufferedWriter.newLine();
bufferedWriter.flush();

And to recieve messages over socket:

private Socket socket;
private ServerSocket serverSocket;
private BufferedReader bufferedReader;

socket = this.serverSocket.accept();
this.bufferedReader = new BufferedReader(new InputStreamReader(socket.getInputStream()));
clientMessage = bufferedReader.readLine();
JSONObject obj = new JSONObject(clientMessage);

3.2 Registration of peers on the network

After the network is created, the first step that has to be done, is to make sure that peers can register themselves. To register, peer is sending the registration object message to all ports that are available. On receiving the message that a new peer wants to connect with a port, that port is added to the network list and as return, all ports that get the registration message then send the list of already connected ports to the newly registered port. Lastly to crown off the registration, logger informs the user, that a new port has been connected with the number. In case that such a number is already connected nothing happens except the logger informing the user that the port number has already been connected to the network.

3.3 Transaction validation

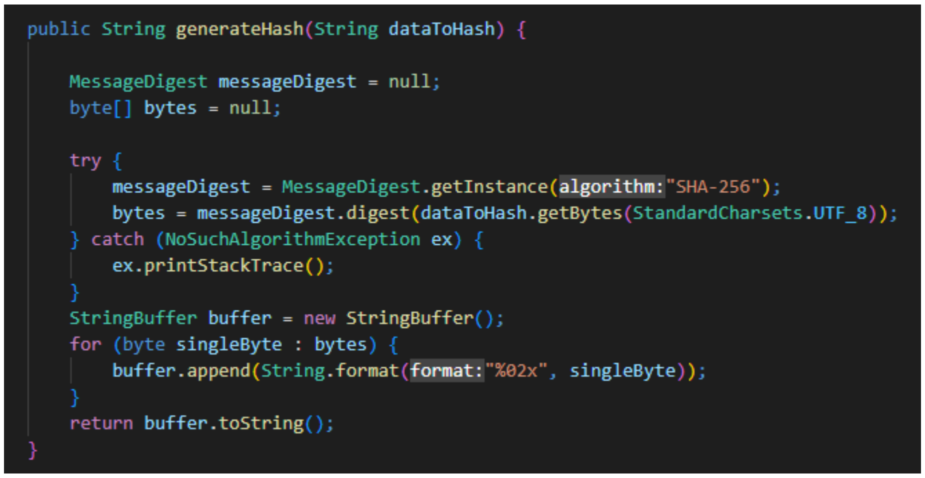
Not every transaction can be sent on the blockchain. To filter out the transactions, the books are checked for the ISBN number. After trying to add a new book to the pool, it is being checked if the book with the same ISBN is already added. If it is already on the blockchain, the message is dismissed and logger informs the user that the transaction is invalid.

3.4 Transaction pool

All ports can add books, by adding books a new message is sent to all connected ports with the book. Every port then has the new transaction added to a pool of ItemTransaction objects. The idea is, that on every forth transaction, a new block is created. To keep count of the number of transactions until a certain point in time, ports can check in the transaction pool how many items are saved. Every time a port sends a transaction, the same port becomes a message back, with the number of transactions saved in the pool. If the number after the transaction is four, then the items from the pool are added to a new block and the pool has to be cleared so it can collect four new transactions before creating new blocks.

3.5 SHA-256 encription

To make the blockchain as safe as possible, for every block created, the cryptographic hash function is called, to created the cryptographic value of the objects in the block. All the String information of the object are concatenated together, and on that, the cryptographic value is determined.
The hash function is defined as a easy to determine value, which is very hard to invert [6] as it has a fixed length of the hash value. Hardly ever there can be two values with the same sha-256 cryptographic value generated for something. In the case of the implementation, digest methods are used to create the UTF_8 value. The size of the cryptographic value is, as the name sais 256 bit long.

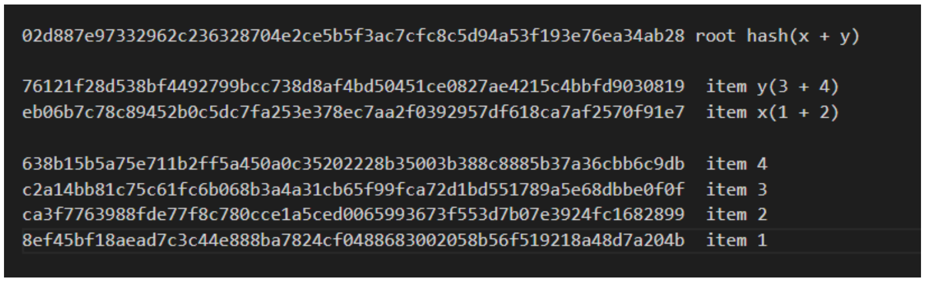


```java
public String generateHash(String dataToHash) {

    MessageDigest messageDigest = null;
    byte[] bytes = null;

    try {
        messageDigest = MessageDigest.getInstance(algorithm:"SHA-256");
        bytes = messageDigest.digest(dataToHash.getBytes(StandardCharsets.UTF_8));
    } catch (NoSuchAlgorithmException ex) {
        ex.printStackTrace();
    }
    StringBuffer buffer = new StringBuffer();
    for (byte singleByte : bytes) {
        buffer.append(String.format(format:"%02x", singleByte));
    }
    return buffer.toString();
}
```

Figure 3: Hashing function snippet

3.6 Merkle tree

The merkle tree ensures that all data in a block is same on every peers end. On creating a block, the four transactions, that are going to be created in the block, first get their own hash number. Then, by combining left and right nodes(transactions) new hash values are created until there is only one root node and it's hash value becomes the current hash for the block. In that way, peers can ensure that the block is same on everyone's end. To know if there was some malicious intend on some peer, that can be easily seen by the root hash value[7]. If the hash number is not same on every peer, that means that one of the peers has tried to manipulate data, that is on the block.
In this implementation, by creating the hash value, the current hash, the previous hash, the timestamp and the transaction object in string are concatenated and the hash value is created from that value.

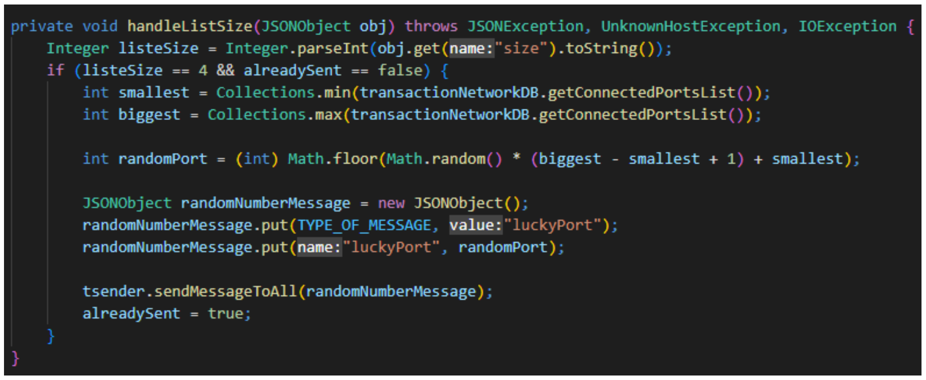


```
02d887e97332962c236328704e2ce5b5f3ac7cfc8c5d94a53f193e76ea34ab28 root hash(x + y)

76121f28d538bf4492799bcc738d8af4bd50451ce0827ae4215c4bbfd9030819  item y(3 + 4)
eb06b7c78c89452b0c5dc7fa253e378ec7aa2f0392957df618ca7af2570f91e7  item x(1 + 2)

638b15b5a75e711b2ff5a450a0c35202228b35003b388c8885b37a36cbb6c9db  item 4
c2a14bb81c75c61fc6b068b3a4a31cb65f99fca72d1bd551789a5e68dbbe0f0f  item 3
ca3f7763988fde77f8c780cce1a5ced0065993673f553d7b07e3924fc1682899  item 2
8ef45bf18aead7c3c44e888ba7824cf0488683002058b56f519218a48d7a204b  item 1
```

Figure 4: Calculation of the root hash - Merkle tree

3.7 Proof of Authority (PoA)

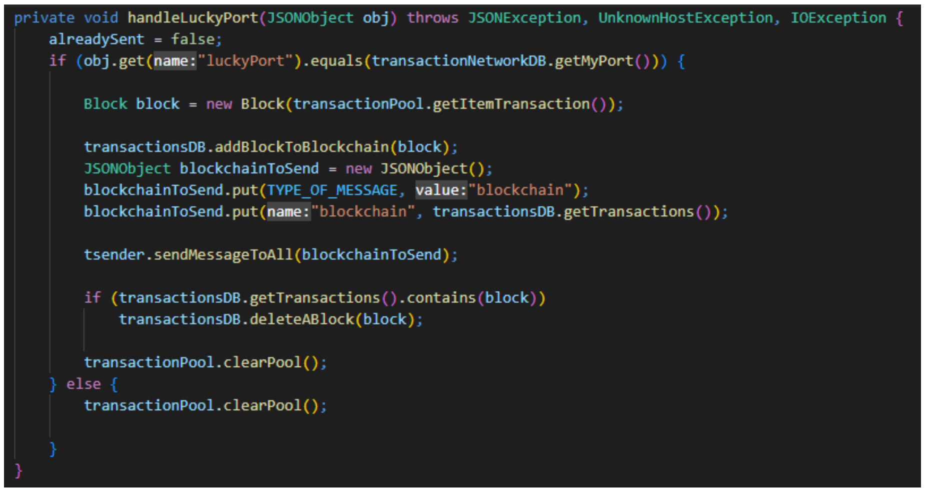
In case of the Proof of Authority, in this implementation it is handled when a peer sends the fourth transaction. Every time a peer sends a transaction/adds book to the pool, to all connected peers, all other peers in return send back the size of the pool of items already saved. That peer checks if the size is already four. If it is not, nothing happens. In case that the pool size is 4, that peer gets to choose the peer that is going to create the block. The peer is selected by generating a random number from the ports list database. Then the port that is the ”lucky” port is being sent to all peers in the network.



```
private void handleListSize(JSONObject obj) throws JSONException, UnknownHostException, IOException {
    Integer listeSize = Integer.parseInt(obj.get(name:"size").toString());
    if (listeSize == 4 && alreadySent == false) {
        int smallest = Collections.min(transactionNetworkDB.getConnectedPortsList());
        int biggest = Collections.max(transactionNetworkDB.getConnectedPortsList());

        int randomPort = (int) Math.floor(Math.random() * (biggest - smallest + 1) + smallest);

        JSONObject randomNumberMessage = new JSONObject();
        randomNumberMessage.put(TYPE_OF_MESSAGE, value:"luckyPort");
        randomNumberMessage.put(name:"luckyPort", randomPort);

        tsender.sendMessageToAll(randomNumberMessage);
        alreadySent = true;
    }
}
```

Figure 5: Generating the lucky port if the pool size is already 4

After the peers check, if the lucky port is indeed themeselve, they get to handle the block creation, addition of the block to the blockchain and sending the blockchain database to all connected ports in the database.



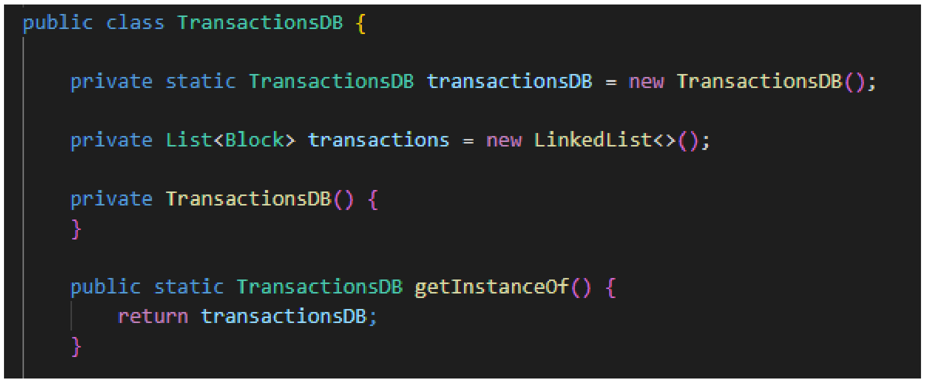
```
private void handleLuckyPort(JSONObject obj) throws JSONException, UnknownHostException, IOException {
    alreadySent = false;
    if (obj.get(name:"luckyPort").equals(transactionNetworkDB.getMyPort())) {

        Block block = new Block(transactionPool.getItemTransaction());

        transactionsDB.addBlockToBlockchain(block);
        JSONObject blockchainToSend = new JSONObject();
        blockchainToSend.put(TYPE_OF_MESSAGE, value:"blockchain");
        blockchainToSend.put(name:"blockchain", transactionsDB.getTransactions());

        tsender.sendMessageToAll(blockchainToSend);

        if (transactionsDB.getTransactions().contains(block))
            transactionsDB.deleteABlock(block);

        transactionPool.clearPool();
    } else {
        transactionPool.clearPool();

    }
}
```

Figure 6: Generating a block and adding to the blockchain

3.8 Singleton pattern

The databases that are being used in this implementation are simple linked lists in java. With linked lists it is confirmed that all data in the lists are going to have the same order. Having the same order in the list plays a important part in the usability of the blockchain. In every kind of blockchain, the order of the blocks and data inside the blocks should have the same order on every peer.

To be able to use the same database class which contains the list of objects in every class, singleton pattern is being used. Basically, singleton classes creates only one instance of the class, and the same instance of the class is being used in every other class that has to use the database.



```
public class TransactionsDB {

    private static TransactionsDB transactionsDB = new TransactionsDB();

    private List<Block> transactions = new LinkedList<>();

    private TransactionsDB() {
    }

    public static TransactionsDB getInstanceOf() {
        return transactionsDB;
    }
```

Figure 7: Singleton pattern

Every class that needs to get data from the database can just use the instance of the TransactionsDB class as followed: private TransactionsDB transactionsDB = TransactionsDB.getInstanceOf();

4 Implementation

In the Implementation of the private blockchain project there are some steps that are followed. The project is done with using the following:

- Visual Studio Code as editor

- Java version 17.0.7

- Maven

- Spring Boot

- SHA-256

- JSON

- Socket

The implementation is based on a usecase of a library, where important books are stored. First of all, peers have to connect with each other, so that they can communicate.
For every communication between peers, which are client and server at the same time, different kind of message is used. The message kinds will be explained later on. After the registration, any peer can add Books, which contain the author name, book name, publishing date and ISBN number, which is unique in the blockchain. After 4 books are added, one lucky port number is generated, by the last peer that has made a transaction(added a book). The lucky port which is determined takes care of creating a block which is then mined on the blockchain. This brief explanation should help understand the process of how the blockchain basically works.
Three separate instances are opened with three different ports in this case instance 1 Spring project is running on port 401 and the socket is running on port 301, the second instance is on port 402, sending messages from 302 and the third instance on port 403 and the socket port 303. To run the instance, the BlockchainApplication.java class has to be run, and the client line interface(CLI) is open automatically for user to chose from 4 different options. User has to register the port first so that it can be used to adding items and displaying the blockchain data.

4.1 How to run the Project

To run this project, it has to be exported from the zip file. After extracting the project, it has to be imported into a workspace on an editor. The project is written in Visual Studio Code, but any other editor is also good. After extracting three folders have to be made for three different instances of the project or three different peers/ports. All three instances have to be open in different editors and different workspaces. After importing the three instances, little editing has to be done to two of the three open workspaces.

First of all, in the application.properties, which is placed under /.../Project/ Implementation/blockchain/src/main/resources/application.properties, the port has to be changed from 401 to 402 in one and 403 in the other workspace. So now all three workspaces have different ports in the application.properties which are 401, 402 and 403. Without that editing, spring boot can not run the instance on the same port because it is already taken.

Furthermore, the classes CLI and CLIHandler also need editing with the ports. The Variable myPort in both classes have to be changed for two of three open workspaces. One of them has to be 302 and the second 303, and so for both open workspaces.

After that, all three workspaces have to be run on the BlockchainApplication.java because that is the main class of the implementation. Then, there are three windows with running CLI, one for 401 one 402 and one for 403. On all three CLI's, the user has to register the peers with the option number 1. After successfully registering on all three peers, user can add books from any of the three peers, and they will be sent to all other ports. On hitting 4 added books, a block is created, and all three peers will be informed, that it has been created. With the command 3 "Print the Blockchain", on any of the three instances, user can see the blockchain as shown in Figure 14

4.2 Package - Objects

There are two classes that represent the objects in this project. The first one beeing the ItemTransaction. This class is basically the Book that is being added. It contains a author name, book name, publishing date, ISBN and port. The ISBN is unique and after a peer tries adding a book to the pool it is checked, if a book with the same ISBN is already mined on the blockchain or added to the current pool of items(books). The port is added to keep track which port has added which book. That fulfills the use-case where someone searches for a particular book in the blockchain and can see where that book is(who added it)

The second object class is Block. This class is the main Object class which is mined on the blockchain. It is created with the hash number, the previous hash, the list of the transaction items. Here on creating a block with the list of the items, merkle tree is created(which will be later explained) and the hash will be determined.

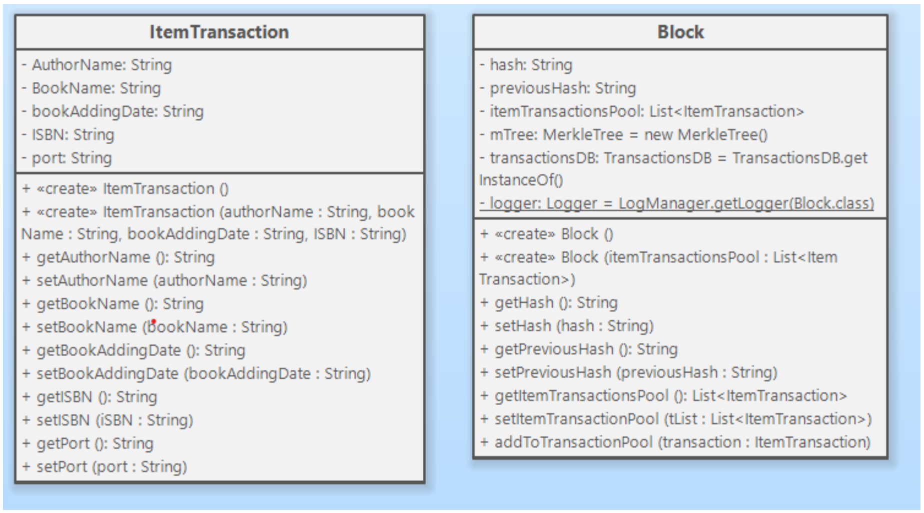

ItemTransaction
- AuthorName: String
- BookName: String
- bookAddingDate: String
- ISBN: String
- port: String
+ «create» ItemTransaction ()
+ «create» ItemTransaction (authorName : String, bookName : String, bookAddingDate : String, ISBN : String)
+ getAuthorName (): String
+ setAuthorName (authorName : String)
+ getBookName (): String
+ setBookName (bookName : String)
+ getBookAddingDate (): String
+ setBookAddingDate (bookAddingDate : String)
+ getISBN (): String
+ setISBN (iSBN : String)
+ getPort (): String
+ setPort (port : String)
Block
- hash: String
- previousHash: String
- itemTransactionsPool: List<ItemTransaction>
- mTree: MerkleTree = new MerkleTree()
- transactionsDB: TransactionsDB = TransactionsDB.getInstanceOf()
- logger: Logger = LogManager.getLogger(Block.class)
+ «create» Block ()
+ «create» Block (itemTransactionsPool : List<ItemTransaction>)
+ getHash (): String
+ setHash (hash : String)
+ getPreviousHash (): String
+ setPreviousHash (previousHash : String)
+ getItemTransactionsPool (): List<ItemTransaction>
+ setItemTransactionPool (tList : List<ItemTransaction>)
+ addToTransactionPool (transaction : ItemTransaction)

Figure 8: Package objects class diagram

4.3 Package - Databases

To normally use the blockchain in this Implementation for databases, normal classes are implemented that are created with singleton pattern.

The first database class is the transaction network DB. In this database the list of the ports, which are connected is saved. For the easy use of the messages, the port number of the current port is also saved. Having all the ports saved makes it possible to send messages to all connected ports. After the registration of a port, the database is updated with the new port that has joined the communication. So that the port, when it sends messages to all ports in the list, does not send the message to itself, the current port is also saved and the messages are sent to all except the current port.

The next database is the transaction pool. So that the peers have an overview how many items are added before creating a block, all the items are first added to this database which is the pool of items. it has the list of transaction items.

The last and the most important database is the blockchain itself. It contains a list of Blocks that are mined to the blockchain after the transaction pool database accumulates 4 different transactions, which are again added books by peers.

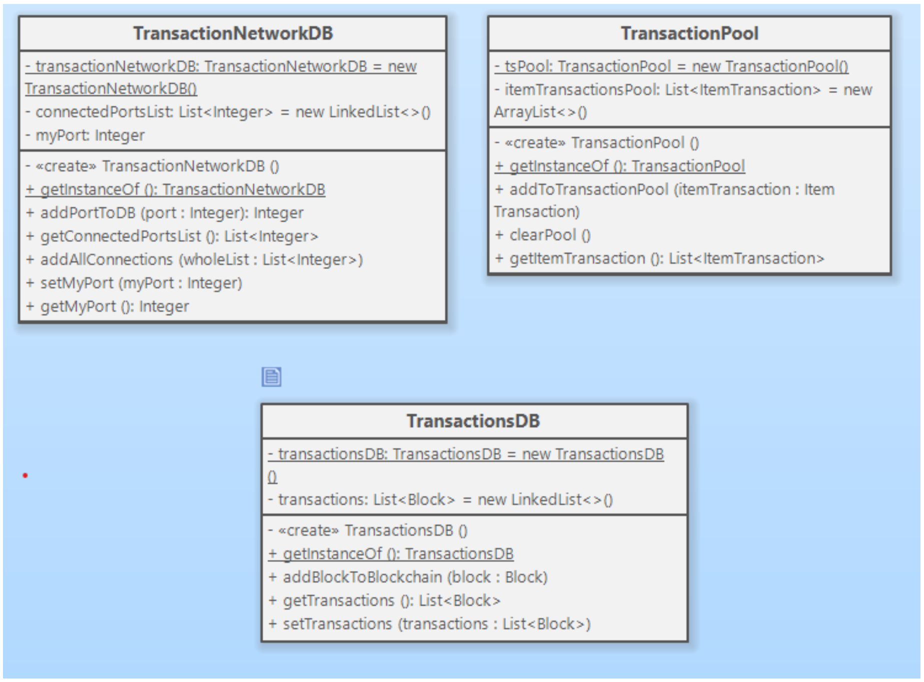

TransactionNetworkDB
- transactionNetworkDB: TransactionNetworkDB = new TransactionNetworkDB()
- connectedPortsList: List<Integer> = new LinkedList<>()
- myPort: Integer
- «create» TransactionNetworkDB ()
+ getInstanceOf (): TransactionNetworkDB
+ addPortToDB (port : Integer): Integer
+ getConnectedPortsList (): List<Integer>
+ addAllConnections (wholeList : List<Integer>)
+ setMyPort (myPort : Integer)
+ getMyPort (): Integer
TransactionPool
- tsPool: TransactionPool = new TransactionPool()
- itemTransactionsPool: List<ItemTransaction> = new ArrayList<>()
- «create» TransactionPool ()
+ getInstanceOf (): TransactionPool
+ addToTransactionPool (itemTransaction : Item Transaction)
+ clearPool ()
+ getItemTransaction (): List<ItemTransaction>
TransactionsDB
- transactionsDB: TransactionsDB = new TransactionsDB ()
- transactions: List<Block> = new LinkedList<>()
- «create» TransactionsDB ()
+ getInstanceOf (): TransactionsDB
+ addBlockToBlockchain (block : Block)
+ getTransactions (): List<Block>
+ setTransactions (transactions : List<Block>)

Figure 9: Package databases class diagram

4.4 Package - Merkle Tree

To hash the new block that is created it has to go through these classes. After the pool is filled with 4 transaction items a merkle tree is created. The creating of the merkle tree is shown here:
The merkle tree is built from Nodes. Every single part shown in this photo are nodes. Class nodes is made of left node, right node and the hash number which is created.
The L1-L4 are the items/nodes that are being saved in the transaction pool. First of all the hash number from the 4 items are generated. Now the 4 items have their own hash code. After that, left and right node are taken and combined and a new hash number is generated. Like that every step left and right node are combined until there is only one node left. That node is then the final hash number form where the block gets the hash.
The hashing is done by using the MessageDygest function. The data are encripted using the SHA-256 enciption.

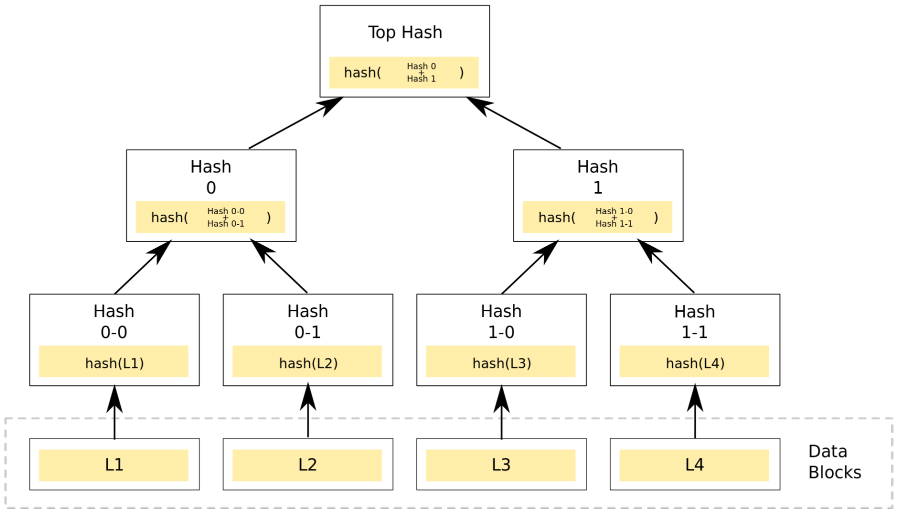
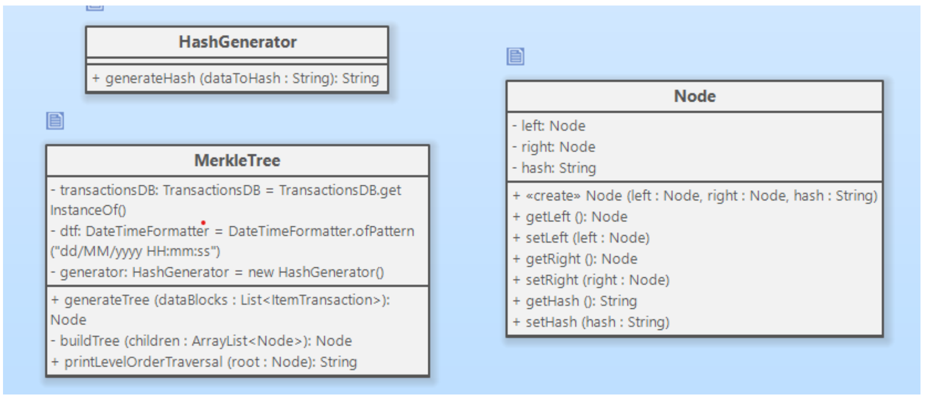


Figure 10: Merkle tree[12]



HashGenerator

+ generateHash (dataToHash : String): String

MerkleTree

- transactionsDB: TransactionsDB = TransactionsDB.getInstanceOf()
- dtf: DateTimeFormatter = DateTimeFormatter.ofPattern("dd/MM/yyyy HH:mm:ss")
- generator: HashGenerator = new HashGenerator()

+ generateTree (dataBlocks : List<ItemTransaction>): Node
- buildTree (children : ArrayList<Node>): Node
+ printLevelOrderTraversal (root : Node): String

Node

- left: Node
- right: Node
- hash: String

+ «create» Node (left : Node, right : Node, hash : String)
+ getLeft (): Node
+ setLeft (left : Node)
+ getRight (): Node
+ setRight (right : Node)
+ getHash (): String
+ setHash (hash : String)

Figure 11: Package merkleTree class diagram

4.5 Package - Blockchain

The main Spring Boot class BlockchainApplication runs the whole project on the port which is declared in the application.properties. The only function besides running the instance is running the client line interface(CLI) for the user.
The CLI expects the user to input the chosen option that they want to do. They can choose between 1. registering the peer which sends the port to all other ports. 2. adding a book to the library which sends the Transaction Item

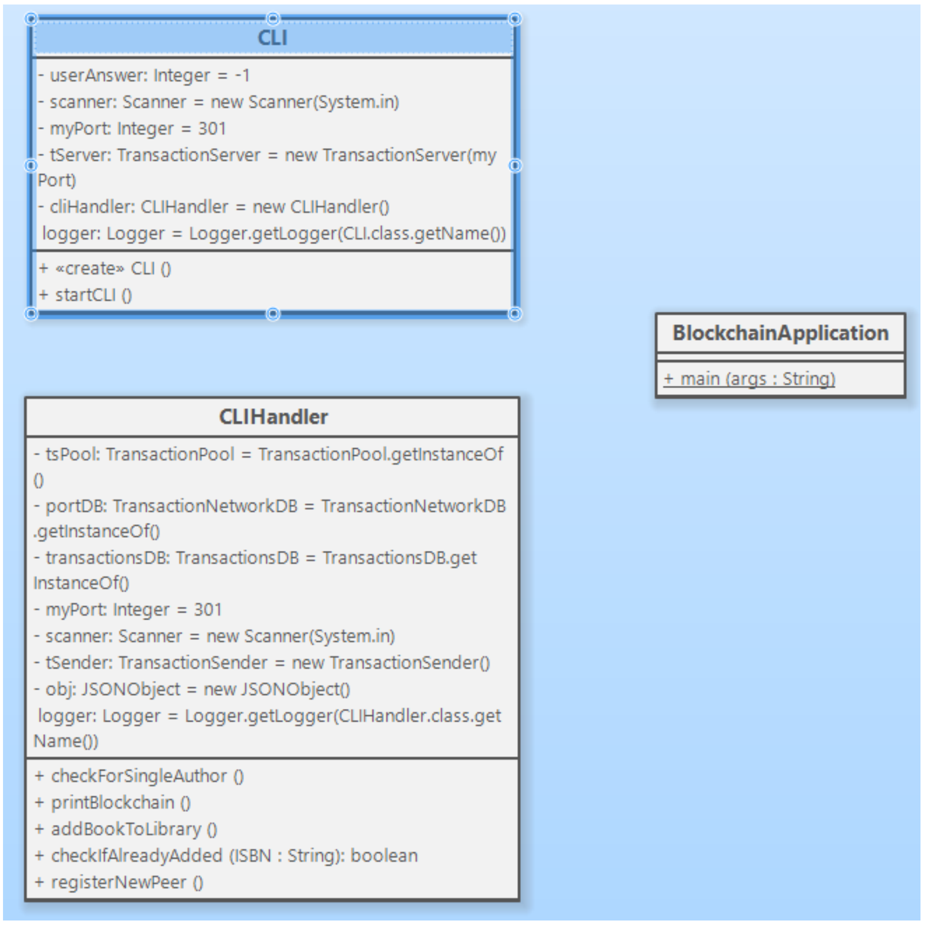
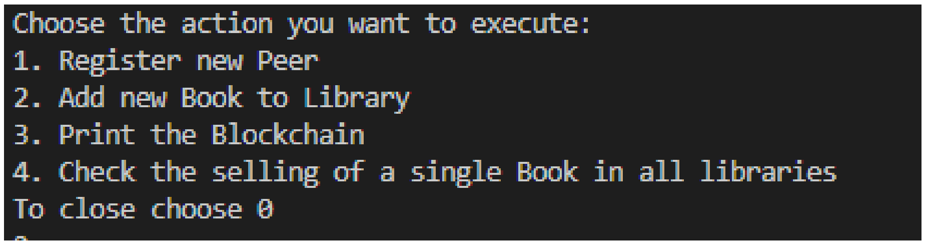

CLI
- userAnswer: Integer = -1
- scanner: Scanner = new Scanner(System.in)
- myPort: Integer = 301
- tServer: TransactionServer = new TransactionServer(my Port)
- cliHandler: CLIHandler = new CLIHandler()
logger: Logger = Logger.getLogger(CLI.class.getName())
+ «create» CLI ()
+ startCLI ()
BlockchainApplication
+ main (args : String)
CLIHandler
- tsPool: TransactionPool = TransactionPool.getInstanceOf ()
- portDB: TransactionNetworkDB = TransactionNetworkDB .getInstanceOf()
- transactionsDB: TransactionsDB = TransactionsDB.get InstanceOf()
- myPort: Integer = 301
- scanner: Scanner = new Scanner(System.in)
- tSender: TransactionSender = new TransactionSender()
- obj: JSONObject = new JSONObject()
logger: Logger = Logger.getLogger(CLIHandler.class.get Name())
+ checkForSingleAuthor ()
+ printBlockchain ()
+ addBookToLibrary ()
+ checkIfAlreadyAdded (ISBN : String): boolean
+ registerNewPeer ()

Figure 12: Package blockchain class diagram

to other ports that are connected. 3. showing the blockchain and 4. searching the blockcahin for a single author. When the user enteres 0 the instance is then terminated. The handling of the user input is done in CLIHandler class.

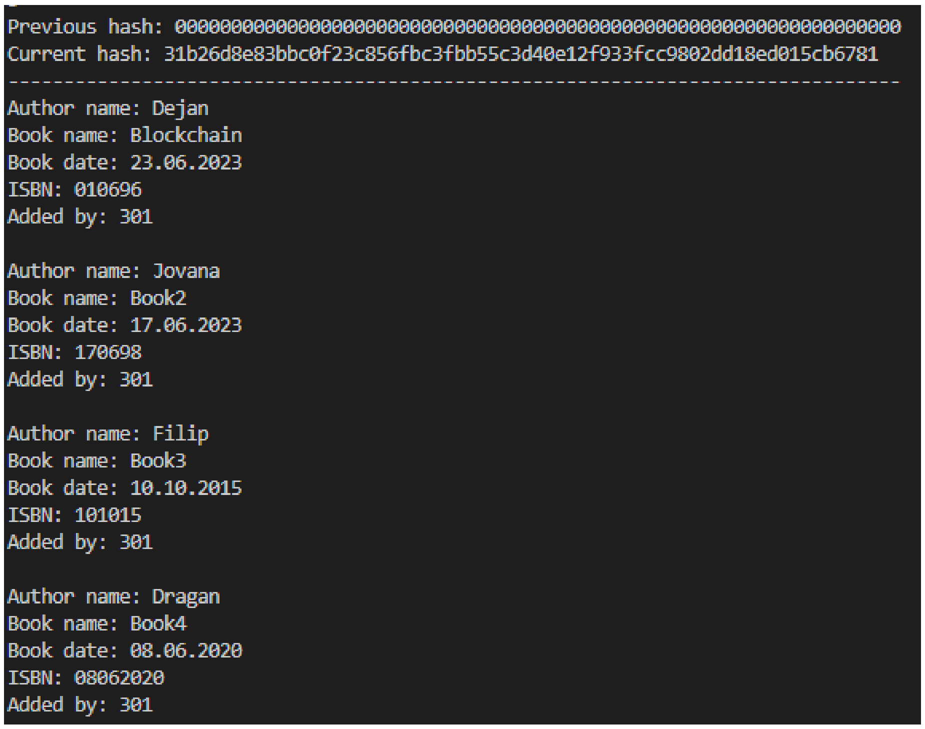


```
Choose the action you want to execute:
1. Register new Peer
2. Add new Book to Library
3. Print the Blockchain
4. Check the selling of a single Book in all libraries
To close choose 0
```

Figure 13: CLI

The peer registration adds and sets the current port to the Ports database and creates a json object with the key ”TypeOfMessage”(which is same for every kind of message that is sent) and value ”reg” which is used for the registration messages. The second part of the json object is key - ”port” and the value is the current port from that instance. Json object is then sent to all other connected ports. The sending and recieving of the messages is handled by the classes Trasnaction Sender and Transaction Server which will be explained later on. After successful registration a logging message is written
INFO 15484 — [Thread-2] a.a.u.b.trasactionP2P.TransactionServer : New Peer connected with port: 302!

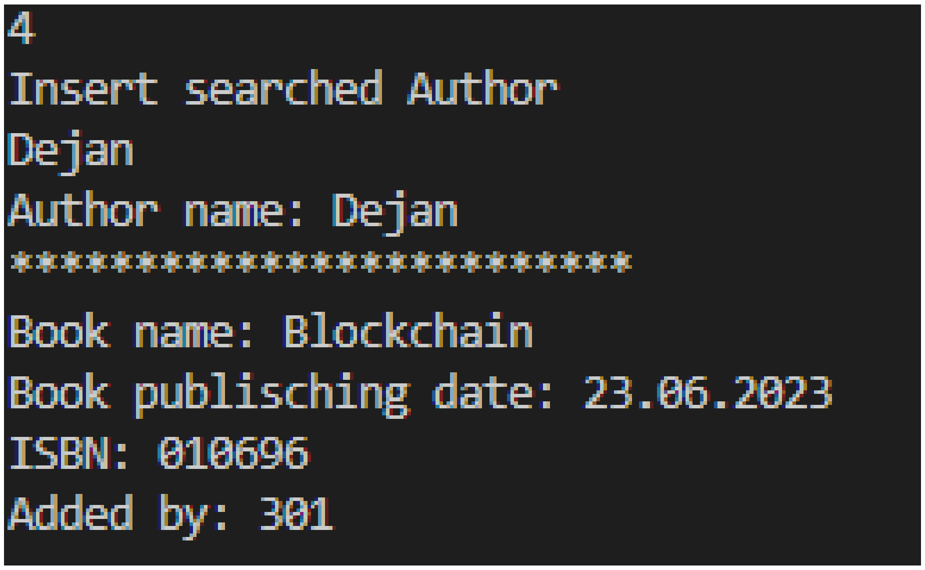
If a port is already connected an infomessage is written.
INFO 8572 — [Thread-2] a.a.u.b.trasactionP2P.TransactionServer : Peer with port: 301 is already connected!
When user chooses to add a book handler function asks the user to add the book author name, the book name, publishing date and the ISBN. At this point it needs to be checked, if a book with the same ISBN is already added in the blockchain and in the pool of transaction items. If the ISBN is not unique, the input of the message is not accepted and the logging message
”INFO 15484 — [main] a.a.u.blockchain.blockchain.CLIHandler : Book with same ISBN already added.”
If it is unique the TransactionItem object and the json object are created. the json object now contains the TypeOfMessage ”transaction”, sends the following data in the json object besides the type: author name, book name, publishing date, ISBN and the port from which the book i added. With that information, we can later determine which peer added the book. That json object is then sent to all connected ports which are saved in the ports database.
The print blockchain function shows the blockchain as shown in the picture below.



```
Previous hash: 0000000000000000000000000000000000000000000000000000000000000000
Current hash: 31b26d8e83bbc0f23c856fbc3fbb55c3d40e12f933fcc9802dd18ed015cb6781
--------------------------------------------------------------------------------
Author name: Dejan
Book name: Blockchain
Book date: 23.06.2023
ISBN: 010696
Added by: 301

Author name: Jovana
Book name: Book2
Book date: 17.06.2023
ISBN: 170698
Added by: 301

Author name: Filip
Book name: Book3
Book date: 10.10.2015
ISBN: 101015
Added by: 301

Author name: Dragan
Book name: Book4
Book date: 08.06.2020
ISBN: 08062020
Added by: 301
```

Figure 14: Bockchain printed out

The previous hash gets the value of the 0 because there are no previous blocks. When another block is created the new blocks previous hash becomes the current hash from the block from the picture.
The last option is to search for a single author in the blockcahin and see all books that one author has published. User has to input the author name and gets the list of all books from the searched author.



```
4
Insert searched Author
Dejan
Author name: Dejan
**************************
Book name: Blockchain
Book publisching date: 23.06.2023
ISBN: 010696
Added by: 301
```

Figure 15: List of books from a single author

4.6 Package - transactionP2P

In this package the whole communication between the peers is happening. For the communication sockets are being used and BufferedWriter to create messages that are transferred over the sockets. The TransactionSender class is for sending messages over sockets. There are different methods for sending the messages. When sending the registration, not only is the port from the peer that is connecting being sent, but also the list of all connected ports until that moment. That is happening because new peers that connect have to get all the ports that are already connected up until that point of time. Peers are also sending transactions to all connected peers. To check how many items there are in the pool, and to see which was the last peer to have added the forth book the size of the pool list is also being sent. That information is just sent to the port which is creating a transaction. After knowing which port the last one to add a transaction to the pool is, that one port is randomly selecting one of the connected ports a as ”lucky” port which will do the mining. That random port is being sent to all connected ports also.
The most important class where most of the functionality is happening is the TransactionServer class. The transaction server is accepting all messages that are being sent to the port over socket. The messages are all sent as json objects which are later on parsed and used. Transaction server is started on the current port and has a run function that is running a thread, which accepts the messages. There are six different types of messages that are being sent. Every message type handles different parts of the conversation in the blockchain. The message types are: reg, lis, transaction, listSize, luckyPort and blockchain.

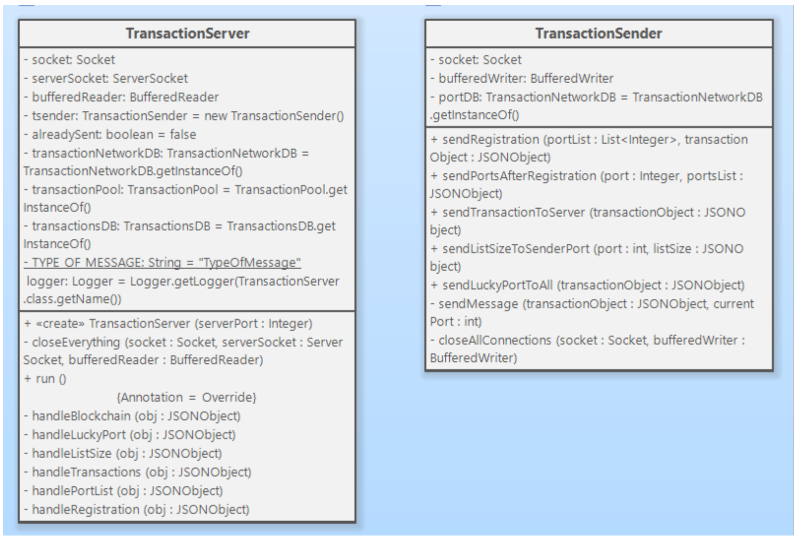

TransactionServer
- socket: Socket
- serverSocket: ServerSocket
- bufferedReader: BufferedReader
- tsender: TransactionSender = new TransactionSender()
- alreadySent: boolean = false
- transactionNetworkDB: TransactionNetworkDB = TransactionNetworkDB.getInstanceOf()
- transactionPool: TransactionPool = TransactionPool.get InstanceOf()
- transactionsDB: TransactionsDB = TransactionsDB.get InstanceOf()
- TYPE_OF_MESSAGE: String = "TypeOfMessage"
logger: Logger = Logger.getLogger(TransactionServer .class.getName())
+ «create» TransactionServer (serverPort : Integer)
- closeEverything (socket : Socket, serverSocket : Server Socket, bufferedReader : BufferedReader)
+ run ()
{Annotation = Override}
- handleBlockchain (obj : JSONObject)
- handleLuckyPort (obj : JSONObject)
- handleListSize (obj : JSONObject)
- handleTransactions (obj : JSONObject)
- handlePortList (obj : JSONObject)
- handleRegistration (obj : JSONObject)
TransactionSender
- socket: Socket
- bufferedWriter: BufferedWriter
- portDB: TransactionNetworkDB = TransactionNetworkDB .getInstanceOf()
+ sendRegistration (portList : List<Integer>, transaction Object : JSONObject)
+ sendPortsAfterRegistration (port : Integer, portsList : JSONObject)
+ sendTransactionToServer (transactionObject : JSONO bject)
+ sendListSizeToSenderPort (port : int, listSize : JSONO bject)
+ sendLuckyPortToAll (transactionObject : JSONObject)
- sendMessage (transactionObject : JSONObject, current Port : int)
- closeAllConnections (socket : Socket, bufferedWriter : BufferedWriter)

Figure 16: Package transactionP2P class diagram

4.6.1 Message type: reg

The first message type is ”reg”. When the server gets a json object in which the messageType ”reg” is, the registration is handled. The port, which is also part of the json object is parsed and added to the ports database. In case that the port is already in the database, a logger will inform, that the peer with the number is already connected, if not then it is going to inform the user that a new port with the port number has successfully been connected. After the connection, all already connected ports send a new json object with the message type lis and the ports from the database, to the one port that is newly connected.
The reg json objects looks like this: { ”TypeOfMessage”: ”reg”, ”port”: ”301” }

4.6.2 Message type: lis

When the message type is lis the server is accepting the json object with the list of ports. This happens because when a new port connects, it needs to get the list of the already connected ports to be able to transfer messages to these ports. The ports list is parsed from the json object and saved to the ports database in the current(newly connected) port.
The lis josn object looks like this: { ”TypeOfMessage”: ”lis”, ”portList”: [301, 302, 303] }

4.6.3 Message type: transaction

Message type transaction handles the accepting of the transaction, as said in this use-case the adding of new books. The json object is accepted and parsed. A new transaction item object is created with the author name, book name, publishing date ISBN and the port, to know which port added the transaction. The new object is then added to the item pool database. To know when the size of the Item pool is 4, the item pool list size from the database is sent after every transaction only to the port, which is sending the transaction. As such, the peer knows if the pool is ready to be created in a block and mined to the blockchain.
The transaction json object looks like this: { "TypeOfMessage": "transaction", "authorName": "Dejan", "bookName": "Blockchain", "date": "23.06.2023", "ISBN": "230623", "port": "301" }

4.6.4 Message type: listSize

ListSize is the message type, which checks if the pool size is 4. When the pool size is 4, that means that the peer that added the last book has the task, to generate a random port which is going to do the block creating, mining on the blockchain and sending the blockchain to all other ports. After generating the "lucky port", a new message type is being created which is "luckyPort" with the number of the port. That new object is then sent to all connected ports.
The listSize json object looks like this: { "TypeOfMessage": "listSize", "size": "4" }

4.6.5 Message type: luckyPort

All connected ports get the message with the luckyPort. Only the one which has the same current port does the work in this function. All other ports just have to clear the transaction item pool database. Lucky port creates a Block object with the transaction pool and adds the block object to the blockchain database. The creation of the block object handles the hashing and creating of the merkle tree. After adding the object to the database, a new json object is created to be sent to all connected ports. The new object has the message type "blockchain" and transfers the whole blockchain database to all connected ports. After sending the message, that port also clears the item transaction pool.
The luckyPort json object looks like this: { "TypeOfMessage": "luckyPort", "luckyPort": "301" }

4.6.6 Message type: blockchain

The the whole blockchain database is transferred in this message type. This method is used mainly to parse the message that is received as json.

The blockchain json object looks like this: ”TypeOfMessage”: ”blockchain”, [{ ”hash”: ”0000000000000000000000000000000000000000000000000000000000000000”, ”prevousHash”: ”8a429c4366cc03ed2282fc52876d6fc768b42f5d5e30f775eada075a19dda951”, ”itemTransactionsPool”:

[{ ”authorName”: ”Dejan”, ”bookName”: ”Blockchain”, ”bookAddingDate”: ”23.06.2023”, ”ISBN”: ”230623”, ”port”: ”301” }, { ”authorName”: ”Jovana”, ”bookName”: ”Book1”, ”bookAddingDate”: ”23.05.2023”, ”ISBN”: ”230624”, ”port”: ”301” { }, ”authorName”: ”Filip”, ”bookName”: ”Book2”, ”bookAddingDate”: ”23.03.2023”, ”ISBN”: ”230625”, ”port”: ”302” }, { ”authorName”: ”Dragan”, ”bookName”: ”Book3”, ”bookAddingDate”: ”20.02.2023”, ”ISBN”: ”230621”, ”port”: ”303” }]]

To parse this firstly the outer part needs to be parsed because they are 2 arrays in the message. The outer array is the block. Hash, previous are parsed, and then the inner array with the itemTransactionPool. The itemTransactionPool contains all the items that are in the blockchain. All have the author name, book name, publishing date, ISBN and port. after Setting the new objects locally for the current port, the new object is then saved in the current ports blockchain database.

4.7 Future work

For future work, the database classes, which contain linked lists could be changed to real databases such as SQL databases, or any other type of database. That would change the project, that it would not have the data saved in lists in the instance, but rather in a real database.
There would be a real web page that would be a frontend, where the user would not have to write numbers to choose between options, but have a real user interface where the data can be filled out in fields and sent to the network. That change would add on to the usability and easier access of the project for users.
A new transaction kind could be added, where there could be peers that are able to borrow books from a peer that added it. Therefor, a change to the transaction object would have to be done. There should be a number of books that a peer is adding. So, the peers that are able to borrow books can now, if any of the books is still available. That would add functionality to the blockchain, instead of the now ability for the peers, to create a book and show, that they own the book.
The validation of the books that are being added has to be changed. In case an author has changed his book, adding a new version of it and wants to add the new edition with the same ISBN as the earlier version of it, the author would not be able to add the book. A new functionality for validation could be added such that, on adding a new book the user is asked, if it is a new edition of a book that is already on the blockchain, in case there is a earlier edition, the new edition can be added despite it having the same ISBN as the other one.

The running of the implementation has to be made easier then editing the code to run the project on three different ports. To be made easier and much more accessible, the project could have been made into a executable jar. Then, there would be three executable jar files, running on three different ports and having three different socket port numbers.

5 Conclusion

In conclusion, this implementation presents a good solution for loads of industries that are in search of good security, transparency, and efficient operations. In this project its shown how a simple private blockchain, that's job is to safely stores information in blocks. The blocks cant be accessed from third parties which is assured with its hash numbers and encrypting.
Private blockchains offer controlled access to a network of participants, ensuring data confidentiality while maintaining the integrity and immutability of transactions. It also keeps the blockchain safe from fraud and outside manipulation. The peers are safely communicating with each other.
However, blockchain bring certain problems and challenges into it. The most important part of a blockchain is the scalability. The implementation has to be robust and scalable so that everything can properly function. As a decentralized project, security mechanisms play also important roles.
The benefits of a blockchain can be seen in using it on use-cases such as finance or healthcare, where the data is strictly hidden from third parties and shold be securely stored and protected.
Lastly the private blockchain has a potential to change the way some industries work and store secure data. It makes the trust, transparency, security and many other factors greater for the stakeholders.

References

[1] Home — alexandrialabs.xyz. https://alexandrialabs.xyz. [Accessed 27-Jun-2023].

[2] Po.et is working on building new tools to re-empower freedom of the press using blockchain — journalism.co.uk. https://www.journalism.co.uk/news/po-et-is-working-on-building-new-tools-to-re-empower-freedom-of-the-press-using-blockc s2/a723587/. [Accessed 27-Jun-2023].

[3] Publica — publica.com. https://publica.com. [Accessed 27-Jun-2023].

[4] Dinh, T. T. A., Wang, J., Chen, G., Liu, R., Ooi, B. C., and Tan, K.-L. Blockbench: A framework for analyzing private blockchains. In *Proceedings of the 2017 ACM international conference on management of data* (2017), pp. 1085–1100.

[5] Gervais, A., Karame, G. O., Wüst, K., Glykantzis, V., Ritzdorf, H., and Capkun, S. On the security and performance of proof of work blockchains. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (New York, NY, USA, 2016), CCS '16, Association for Computing Machinery, p. 3–16.

[6] Gilbert, H., and Handschuh, H. Security analysis of sha-256 and sisters. In *ACM Symposium on Applied Computing* (2003).

[7] Mohan, A. P., Gladston, A., et al. Merkle tree and blockchain-based cloud data auditing. *International Journal of Cloud Applications and Computing (IJCAC) 10*, 3 (2020), 54–66.

[8] Nelaturu, K., Du, H., and Le, D.-P. A review of blockchain in fintech: Taxonomy, challenges, and future directions. *Cryptography 6*, 2 (2022).

[9] Pervez, H., Muneeb, M., Irfan, M. U., and Haq, I. U. A comparative analysis of dag-based blockchain architectures. In *2018 12th International Conference on Open Source Systems and Technologies (ICOSST)* (2018), pp. 27–34.

[10] Robertson, S. What is a private blockchain?, Sep 2022.

[11] Team, S. C. scalability, Jun 2023.

[12] Wikipedia contributors. Merkle tree — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Merkle_tree&oldid=1159724280, 2023. [Online; accessed 23-June-2023].

[13] Xiao, Y., Zhang, N., Lou, W., and Hou, Y. T. A survey of distributed consensus protocols for blockchain networks. *IEEE Communications Surveys & Tutorials 22*, 2 (2020), 1432–1465.

[14] Yang, R., Wakefield, R., Lyu, S., Jayasuriya, S., Han, F., Yi, X., Yang, X., Amarasinghe, G., and Chen, S. Public and private blockchain in construction business process and information integration. *Automation in construction 118* (2020), 103276.