UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

Escuela Técnica Superior de Ingeniería de Telecomunicación
Universitat Politècnica de València

Blockchain technology analysis and development of a Java implementation.

DISSERTATION

Degree in Telecommunications Technology and Services Engineering

Author: José Blasco Núñez de Cela

Tutor: Antonio León Fernández

Year 2017-2018

Thank you to those who helped me.
Family, friends and friends who are like family.

Vires in numeris

Abstract

In this project we will focus on the analysis of the Blockchain technology and develop a basic implementation in Java, including all the basic components such as transactions, consensus and a P2P network. Lastly, we will show a use case by developing a Proof of Existence platform with our own Blockchain.

Key words: Blockchain, Java, P2P, PoE

Resumen

En este proyecto nos centraremos en el análisis de la tecnología Blockchain y desarrollaremos una implementación básica en Java, incluyendo todos los componentes básicos como transacciones, consenso y la red P2P. Por último, mostraremos un caso de uso mediante el desarrollo de una plataforma de Prueba de Existencia con nuestra propia Blockchain.

Palabras clave: Blockchain, Java, P2P, PoE

Resum

En aquest projecte ens centrarem en l'anàlisi de la tecnologia Blockchain i desenvoluparem una implementació bàsica en Java, inclosos tots els components bàsics com transaccions, consens i la xarxa P2P. Finalment, mostrarem un cas d'ús mitjançant el desenvolupament d'una plataforma Prova d'existència amb ela nostra propia Blockchain.

Paraules clau: Blockchain, Java, P2P, PoE

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

"I give you a horse for two pigs."

Trading has been an essential issue during the development of humanity. We started trading goods for other goods, and then introduced an intermediary step, such as gold and eventually euros.

This system works just fine, we can buy and sell anything we want as far as we have enough euros. We can trade Euros to dollars to buy things in the USA, to Pounds to buy in the UK or to yen to buy in Japan. But this also comes with some issues.

Blockchain technology offers solutions to problems we didn't know that even existed. With blockchain technology, we can have a decentralised, anonymous, secure network to store transactions of money, goods and ideas. We can use this secure, future-ready technology to create a new economic ecosystem, with new currencies and contracts.

Blockchain technology is used to put aside intermediate steps, avoiding having to rely on banks or even electricity companies, making thus trading more secure and anonymous, not to mention avoiding extra cost in form of money and time.

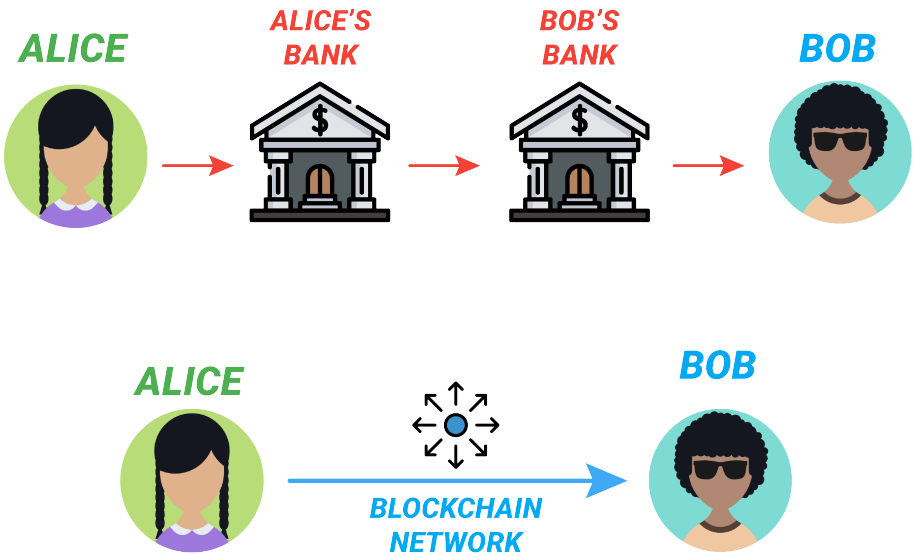

Figure 1.1: Comparison of traditional vs blockchain transaction.

However, money is not the only thing Blockchain is disrupting. Potentially every single industry can adapt Blockchain for their own field of work and be part of this revolution.

Industries such as transport, energy, smart cities, law or even supply chains are slowly exploring this technology and we will use it daily in no longer than five years.

1.1 Goals and distribution of the project

The main goals of this project are to understand how blockchain technology works, and what can it can offer to the society and business of the future. The project is divided in four big chapters.

The first one is an introduction to blockchain and some basic concepts in order to understand how the technology work and be able to comprehend the rest of the project. This is mainly a theoretical part with general concepts that do not relate to any specific blockchain application.

The second and third chapters are where most of the project itself is explained. On the second chapter, we will develop a blockchain in Java from the groundup. On different iter-

ations we will introduce new features step by step and testing and comparing the different versions.

On the third chapter, we will adapt this application to work as a Proof of Existence platform, allowing users to register documents and share them thought the network in a secure and private way, while assuring the integrity of the document.

On the last chapter, we will analyze other blockchain applications aside from PoE, such as cryptocurrencies, IoT technology, smart cities and others.

Planning and introduction of the project	Week 1
Research and introduction to blockchain technology	Weeks 2 & 3
Write chapter 'blockchain basics'	Week 4
Preparation and design of the Java implementation	Week 5
First iteration: core of the blockchain	Weeks 6 & 7
Second iteration: dynamic difficulty	Week 8 (a)
Third iteration: SHA-512 hash	Week 8 (b)
Fourth iteration: users	Week 9
Fifth iteration: transactions	Week 10
Development of a PoE platform	Week 11
Other blockchain applications	Week 12
Formatting and final changes to the memory	Week 13
Preparation of the presentation	Week 14

Table 1.1: Tasks to perform

CHAPTER 2

Blockchain basics

Blockchain is a technology that uses a series of blocks linked between them that can't be edited after they are chained to the network. These blocks can contain any kind of information, but in most of the implementations of this technology, the information stored are the transactions that happen on the network during the time that the block is being built. Therefore, the blockchain works as a decentralized distributed database that synchronizes on every node of the network.

Blockchains can be private (for some business) or public. On private blockchains, the data can be distributed among several places or can be stored just in the company's servers. On the other hand, public networks are open for anyone to see, collaborate and maintain.

The main characteristics of a Blockchain [1] are:

- It is designed to be **distributed and synchronised** across networks.
- You can't just do whatever you want to the data. The types of transactions somebody can carry out are **agreed between participants** in advance and stored in the blockchain according to the design of the network.
- **Immutability of the data**. Once you have agreed on a transaction and recorded it, it can never be changed. You can subsequently record another transaction about that asset to change its state, but you can never hide the original transaction.

2.1 What is a block?

Blocks, as the "Blockchain" name indicates, are the core of the blockchain. Each block contains the information that builds the blockchain. The contents of the block may vary depending on the implementation of the blockchain and its goal. For instance, on a cryptocurrency blockchain, blocks contain the transactions that happen while that block was being built.

The first block of a blochchain is called the "genesis" block, and it usually includes some unimportant info, or a commentary of the developer. Bitcoin's genesis block contained a headline of a newspaper a month earlier than the creation of the network, " *The Times 03/Jan/2009 Chancellor on brink of second bailout for banks*"[2]

Each block is linked to the previous and next one. This is done usually by including the hash of the previous block as part of the current one, therefore if the previous one is modified, it can be detected. This creates a robust system where no one can edit previous blocks in their advantage. New transactions are constantly being processed by miners into new blocks which are added to the end of the chain and can never be changed or removed once accepted by the network.

An example of a bitcoin block structure[3] is the following. We will get into more detail in the bitcoin specific section of the project.

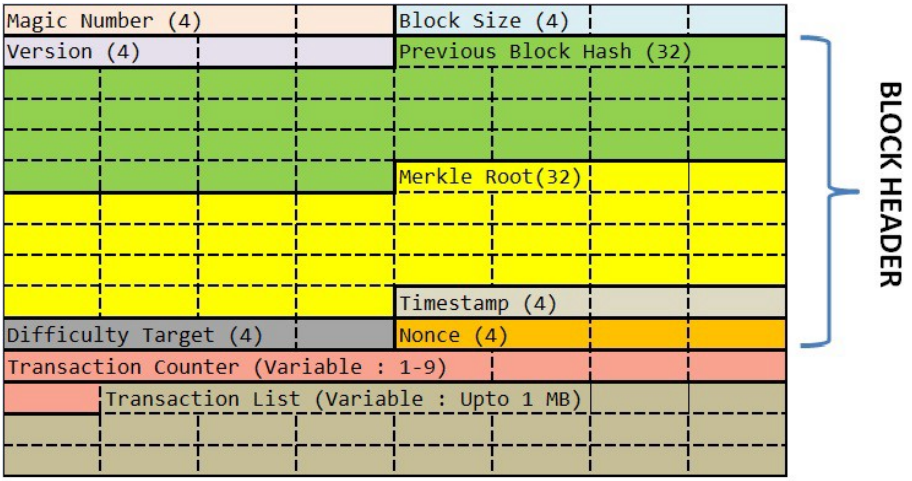

Figure 2.1: Structure of a bitcoin block.

2.2 Achieving consensus on cryptocurrencies

Every cryptocurrency has its own **public blockchain** to store all the transactions that occurred. A proof of work and proof of stake algorithm[4] are different methods (or algorithms) to achieve consensus on which block will be added next to the blockchain.

Proof of work (PoW) requires proof that work of some kind has occurred. In the case of Bitcoin, miners are required to do this work before any of their blocks is accepted by others. In PoW you are as powerful as your hardware is, with respect to the network.

Proof of stake (PoS) requires users that have a high stake at the currency (i.e. hold a lot of coins) to determine the next block. This has a high risk of some party achieving monopoly of the currency but there are several methods to prevent that (we will analyze it later on the project). If you have 5% of the total amount of coins in the network, you will receive 5% of the new tokens created.

The main difference could be summarized in that proof of work requires burning an external resource (mining hardware with a lot of computational power) while proof of stake does not. Proof of work could be criticized that if the profitability (earnings with respect to investment) drops then less people will have incentives to mine thus the security of the system is reduced. Proof of stake could be criticized that since it is free to stake/add new blocks to the blockchain you could use it to stake several similar coins at the same time without any additional effort.

There is a third, less used technology called **Proof of Importance (PoI)**[5]. This technology aims to solve the problems mentioned above, by giving more importance to those who are more valuable to the network, mainly by being more active and trustful, therefore loyal to the network. The first cryptocurrency that used this technology was NEM(XEM).

Examples:

- Bitcoin, Litecoin and many others use the PoW method.
- NXT, BitShares and others use the PoS method.
- Ethereum uses PoW but is scheduled to change to PoS.
- Peercoin uses a combination of PoW and PoS.

This process of achieving consensus is also known as "mining". Mining is the operation of collaborating on the maintenance of the network by confirming transactions and closing blocks that are chained to the network. Mining takes a lot of different forms depending on the consensus technology the network is using.

In PoW, a "miner" (person who mines) uses computational power in order to find a hash of the current block with a defined goal. Usually, this goal is set by the difficulty by defining the hash to start with a number of '0's. To do so, the miner adds a random string to the block in order to change the outcome of the hash. The difficulty is dynamically changed to maintain a consistent time between blocks. The more computational power there is on the network, the more difficult the process is (more '0's at the beginning of the hash).

For example, in an imaginative hashing algorithm, the following hashes occur:

INPUT	HASH OUTPUT
TX1 TX2 TX3	abcdefghijk1234567
TX1 TX2 TX3 random text	00lmnopqrstu987654

Table 2.1: Hashing with an imaginative hashing algorithm

If the target was to get a hash that starts with "00xxxx..." the second input would meet the difficulty and close the block. With this method, the more computing power you have, the more rewards you get.

On PoS, instead of "wasting" energy and computing power on mining, the network closes the blocks automatically every given time (defined by the network) and rewards the miners who host the nodes of the blockchain with a proportional amount of coins with respect of how much they have in their wallets. This way, the more coins you have, the more rewards you get.

As you might have guess, mining is not viable if your resources are small, since you will probably get no reward in PoW and very little in PoS. That's why people join each other in mining pools. A mining pool is a conglomerate of miners that work together to mine the coins, and split the rewards among them. Even though on the long term the rewards a miner gets are similar to what they would earn doing solo-mining, they will get rewards asidually.

2.3 Privacy and security on a Blockchain network

One of the key features of the blockchain technology is the robustness and privacy it offers. This relies in two main ideas, first of all, users are identified by an address and not a name, and secondly by the inherent property of the blockchain to be regulated and protected by its users.

Each user has to be identified on the network, and this is done using addresses. The format of this address is irrelevant, and can take many forms. The important thing is the purpose of the address, sending and receiving transactions inside the blockchain. Addresses are unique and secure. If we had to compare it, we would think this is the equivalent of an IBAN number. However, these addresses are not assigned by any central authority.

Every Blockchain implementation has its own way of designing addresses, but the basic idea is having an address, a public key and a private key. We will get into the details of some cryptocurrencies technology in the next chapter. On private blockchains, each implementation has its own characteristics, so we can't get into details.

Something all implementations of the blockchain have is that they use hashing to secure the blockchain.

2.3.1. What is hashing?

In simple terms, hashing means taking an input string of any length and giving out an output of a fixed length. This means that whether you try to "hash" a single character or a full document, the output of this operation will be a string with the same length. For example, using MD5 (a basic algorithm that is in currently in disuse) we will get the following results.

Every cryptographic hash function has four properties that makes it secure.

INPUT	MD5 HASH OUTPUT
HI	49f68a5c8493ec2c0bf489821c21fc3b
JOSE BLASCO	0a41ed1c3ffd435766021e68ef7409c1
TEXT123	11baab40ab8f2dfe291122dfc56ac9e1

Table 2.2: Hashing with MD5

1. **Fast:** The hash function is quick enough to perform the operation a lot of times per second.
2. **Deterministic:** Every time you perform the operation on the input, the output will be the same.
3. **Collision resistant:** Two different inputs can't have the same output. This is sometimes difficult to accomplish because the hashing output is a fixed length string.
4. **Pre-image resistance:** This means that doing the operation is quick in one way, but unfeasible to do in the other way.

2.3.2. Reverse-engineering the hash of a block

When choosing a hashing algorithm for a blockchain, one of the most important characteristics is the "collision resistance".

This characteristic avoids anyone from creating a fraudulent block and spoof it into the network. Imagine my block A has the following contents:

Block A
Transactions: A sent 50 tokens to B C sent 2 tokens to A
Hash: abcdefghijklmnopqrstuvwz

Block A'
Transactions: A sent 5000 tokens to BAD PERSON B sent 150 tokens to BAD PERSON [random contents to get hash]
Hash: abcdefghijklmnopqrstuvwz

Now imagine someone finds a way to achieve the block A' with the same hash. Since the blockchain is verified using that hash, the attacker can introduce this block into the network after some time and every node will accept it as valid, faking the transaction and hacking the network.

2.3.3. "51% attack"

On public blockchains, whether or not a block is accepted as part of the chain is achieved by consensus as we have already seen. But when consensus is manipulated, a so-called 51% attack happens.

It does not matter if you are using PoW, PoS or any other technology, if a person or a group of people gets control of more than 50% of the power, they can approve or deny the acceptance of any block they want. Imagine that Alice (A) is in control of the blockchain, and Bob (B) wants to send 10 tokens to Alice in order to pay for a bottle of water. Alice can now say that Bob has only paid 5 tokens and refuse to give him the bottle, while taking the 10 tokens. Or even worse, Alice can say that Charlie (C) has transferred to her 500 tokens even if Charlie is unaware of this.

Blockchain networks are aware of this, and even those who could make profit out of it decide not to, since the community would notice and abandon the blockchain, making it useless and the attackers would not get any profit. This was the case of Ghash.io, which briefly exceed 50% of the bitcoin network's computing power in 2014. The pool voluntarily reduced its share and promised to never again reach 40% of the total mining power [6].

We will test this attack on our own cryptocurrency later in the project.

2.4 Forks of a Blockchain

Blockchain is a technology, and every technology evolves. Every implementation is different, they have a different design and a different approach when they are implemented, and every user accepts and works on this design.

This is what gives a blockchain the robustness that we have already explained. Everyone works together on the same way with the same goal and the same parameters, but what happens when someone thinks the network needs a change?

This change can be anything: changing the name, the size of a block, the hashing algorithm... and this is where the problem appears, as some of the users might be against this change.

Forking is the process where a person or a group "clones" the code and technology and develops and implements the technology independently of the original implementation. There are three main types of forks [7]:

1. **Hard fork.**

   A hard fork is a software upgrade that introduces a new rule to the network that isn't backwards-compatible. Hard forks are something like an expansion of the rules. For example, changing block size from 2 MB to 10 MB, since blocks of 5MB are not allowed in the initial version and the transactions on that block would be marked as invalid.

   In this case, if part of the community does not change, a new network must be created separately.

2. **Soft fork.**

   A soft fork, by contrast, is any change that's backward compatible. Say, instead of 2 MB blocks, a new rule might only allow 1 MB blocks.

   Non-upgraded nodes will still see the new transactions as valid (500k is less than 1 MB in this example). However, if non-upgraded nodes continue to mine blocks, the blocks they mine will be rejected by the upgraded nodes. This is why soft forks need a majority of hash power in the network.

   What can go wrong? When a soft fork is supported by only a minority of hash power in the network, it could become the shortest chain and get orphaned by the network. Or, it can act like a hard fork, and one chain can splinter off.

3. **User-activated soft fork.**

   A user-activated soft fork (UASF) is a theoretical idea that explores how a blockchain might add an upgrade that is not supported by those who provide the network's hashing power, for instance to be forced by wallets, exchanges or other network participants.

   Currently this idea is theoretical and has not been implemented.

CHAPTER 3

Developing a blockchain in Java

One of the goals of this dissertation is to discover how a blockchain works, and there is no better way to do so than creating one from scratch. We will start with a simple network to put text into a blockchain, this will allow anyone on the network to place whatever text they want and ensure that it will not be modified while anyone has access to it. Then we will develop different implementations in order to achieve different goals, such as identifying who added the text, or adding transactions, automatically change mining difficulty and implement different clients (host of a blockchain, miner, user...).

Among every programming language there is, Java is probably not the ideal one, but it allows us to apply everything we have learned through the degree, which is one of the key objectives of every dissertation. We aim to use object programming to implement the blockchain, and sockets to communicate among the devices on the network.

The code for every iteration is available on the different branches of the Github repository:

https://github.com/pepebndc/Blockchain-JAVA

3.1 First iteration: core of the blockchain

A blockchain, by definition, is a big P2P network. We design our network based on this principle. With every event, the information is broadcasted through the network. Therefore, every node must have information on every other node. In this iteration, the information stored is the IP address of every node. For now, we are only capable to connect nodes on the same local area network, although if all the clients had a public IP address, the network would work.

The goal of this iteration is to create the foundation for future developments. Therefore, functions such as consensus between the nodes or dynamic changes on the difficulty are not yet implemented. The code for this iteration can be found in:

https://github.com/pepebndc/Blockchain-JAVA/tree/v1–core

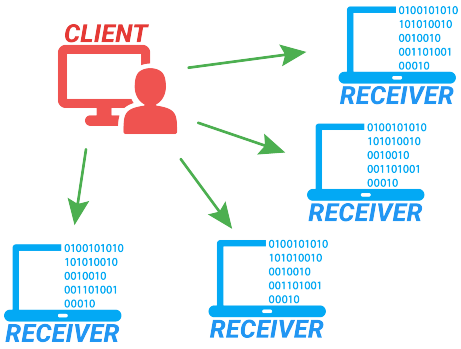

Figure 3.1: Structure of broadcast network.

Since this is the first iteration, and the base for future iterations, we are going to explain the structure of the program.

We are using the following custom objects to perform the operations:

BlockChain Object

The object BlockChain is the biggest object in our program, and contains all the information of the network. It contains:

1. **Name:** The name of the blockchain. It's only used to show it to the user
2. **Difficulty (diff):** The difficulty of the blockchain used to mine the blocks
3. **A list of Blocks:** A list of every block of the blockchain and its contents
4. **Current mining contents:** The contents that are currently being mined and waiting to be included into a block
5. **A list of hosts:** A list containing all the IP addresses of the nodes currently connected to the network

It also has the method *validateChain*, which checks if the contents of the blockchain have been modified, by comparing the hashes of the blocks sequentially.

```
public boolean validateChain() throws NoSuchAlgorithmException,
    UnsupportedEncodingException {
    String[] hashes = new String[this.getChain().size()];
    int length = this.getChain().size();

    Iterator<Block> iterator = this.getChain().iterator();

    while (iterator.hasNext()) {

        Block b = iterator.next();

        String hash = b.getHash();
        int index = b.getIndex();
        String hashPreviousBlock = b.getLastHash();
        long time = b.getTime();
        String data = b.getData();
        int nonce = b.getNonce();

        hashes[index] = hash;
        String newHash = Block.computeHash(index, time, data, hashPreviousBlock,
            nonce);

        if (index != 0) {
            if (index != 1) {
                if (index < (length - 1)) {
                    if (!hash.equals(newHash)) {
                        System.out.println("Different recomputed hash in block " +
                            (index - 1));
                        return false;
                    }

                    if (!hashPreviousBlock.equals(hashes[(index - 1)])) {
                        System.out.println("Different previous hash in block " +
                            (index - 1));
                        return false;
                    }
                }
            }
        }
    }
    return true;
}
```

Block Object

A block contains the information of a single block of the chain, including:

1. **Index:** Number of the block in the chain
2. **Time:** Current time when the block was mined
3. **Data:** Contents of the block
4. **Last Hash:** Hash of the previous block. It is used to verify integrity of the blockchain
5. **Hash:** The hash of the block, stored for informational purposes
6. **Nonce:** Random number that helps getting the hash below a defined value

It also implements the methods *mineBlock* and *computeHash*. *computeHash* provides the MD5 hash of the contents we want to include in the hash of the block (index, time, data, lastHash and nonce), while *mineBlock* performs the mining operations by continuously computing the hash and increasing the nonce until the hash matches the difficulty.

CommandMessage Object

We have designed the whole protocol from scratch, and decided that the best way to transmit information among hosts in the network is to send custom objects filled with the information we share. This object consists of:

1. **Command:** This is the command sent. We will explain the different commands in the next section
2. **Address:** A string used to send the IP address when needed
3. **Blockchain:** A BlockChain object in order to send the whole chain and it's information
4. **Block:** To send a block when needed
5. **Contents:** Used to send other information such as difficulty

List of commands

In this first iteration, the commands are sent using sockets and sending the *CommandMessage* Object with the information needed in each case. When a command has to be sent to all the host in the network, we use an iterator to recursively fetch the different IPs and send the information. The different commands are:

1. **NEW HOST CONNECT:** This message is sent by a new host who wants to connect to the network. The new host can connect to any member of the network with this message, the member will notify the rest of the network of this new host sending a NEW HOST ADD message and responding to the new member with the contents of the Blockchain (NEW HOST ACCEPTED).
2. **NEW HOST ACCEPTED:** This message is sent to the new host and contains the information of the blockchain. This host places the BlockChain received into its Blockchain object.

3. **NEW HOST ADD:** This message is sent to the rest of the members of the network to notify the addition of a new member. This message contains the IP address of the new host and the receiver of the command has to add this IP to its host list.
4. **ADD CONTENT:** Message containing the new contents to be mined. When received, the host updates the mining contents.
5. **DIFF:** Message containing the new difficulty of the network. When received, the host updates the difficulty.
6. **NEW BLOCK:** When a new block is mined, this message transmits it through the network and the other hosts add it to their chain.
7. **DISCONNECT:** When a host disconnects, it sends a signal to the rest of the network and the other nodes take it out of the hosts list.

Executing the program

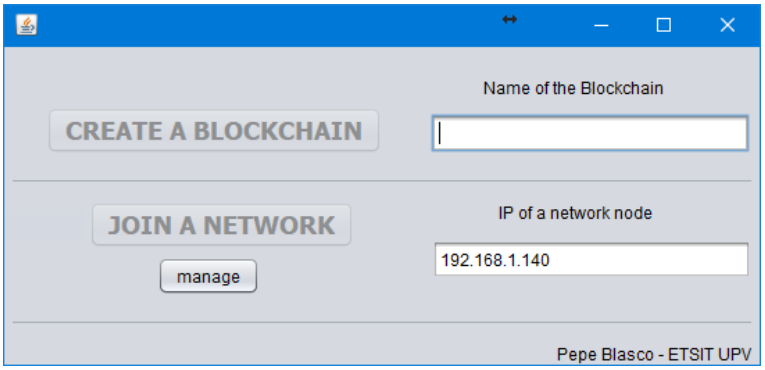

Figure 3.2: Start window.

When launching the program, the user has the option to create a new blockchain network or to join a running one. The buttons are disabled until the user places an input in the corresponding text fields. When creating a blockchain, you need to name it, whereas when connecting to an existing one, you have to place the IP of a host already in the network. After performing one of these actions, the management window is opened (automatically when creating a blockchain, and when clicking on "manage" when connecting to one).

The execution consists of two main components: the mining process and the TCP server.

The **mining process** is a thread where the program is continuously computing the hash of the block and comparing it to the difficulty (number of '0's at the beginning of the hash). Once it matches, the block is added to the blockchain and the network is notified with the *NEW BLOCK* command. This thread also has a "stop" switch built-in, although it is not yet possible to trigger it on the managing window.

On the other hand, the **TCP server** is a thread continuously listening to port 4001 to commands, and managing the reaction to those commands. The *serverManager* creates up to 256 sessions so that it can handle several clients at once. When sending messages to other hosts, we use the **TCP client** where every *CommandMessage* is created in methods where the only input is the IP of the destination (end point). An example of an outgoing connection is the following:

```
public static void sendNewContent(String endPoint) {

    try {

        ObjectOutputStream oo = null;
        Socket so = null;
        try {
            so = new Socket(endPoint, 4001);
            oo = new ObjectOutputStream(so.getOutputStream());
        } catch (IOException e) {
            System.out.println("Problems conecting on TCP");
            System.out.println(e);
        }

        String com = "ADD CONTENT";
        String a = null;
        BlockChain bc = null;
        Block b = null;
        String c = main.TFG.getCurrentMiningContents();

        CommandMessaje command = new CommandMessaje(com, a, bc, b, c);
        oo.writeObject(command);
        oo.flush();
        oo.close();
        so.close();
        System.out.println("new contents messaje sent to " + endPoint+" - "+c);
    } catch (IOException ex) {
        Logger.getLogger(TCPclient.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

Once the Blockchain is connected, you can enter the managing window, where you can interact with the Blockchain and the network. On this window you can see the name of the Blockchain you are connected to at the top, and then the following options are displayed from top to bottom:

1. **ADD NEW CONTENT:** When you enter text on the text field and click *ADD*, the contents of the field are added to the *currentMiningContents* and sent to the rest of the network. This way, the text you insert will be mined and added to the next block.
2. **VALIDATE:** Pressing this button displays a text indicating whether or not the blockchain is valid.
3. **CHANGE DIFF:** It changes the current mining difficulty to the number in the text next to it and broadcasts the change to the network so that everyone has the same difficulty on the next block.
4. **SEE / CHANGE CONTENTS:** Entering a block number shows the contents of that block in the Blockchain, you can then edit those contents and when clicking *CHANGE CONTENTS* the new text will be placed into the contents of the block and the hash will be recomputed.

   This action will not broadcast the changes to the network for safety purposes and will invalidate the blockchain (when clicking on *VALIDATE* it will say "not valid").

Lastly, when the user closes the window, the system automatically sends the *DISCONNECT* message to all the other nodes.

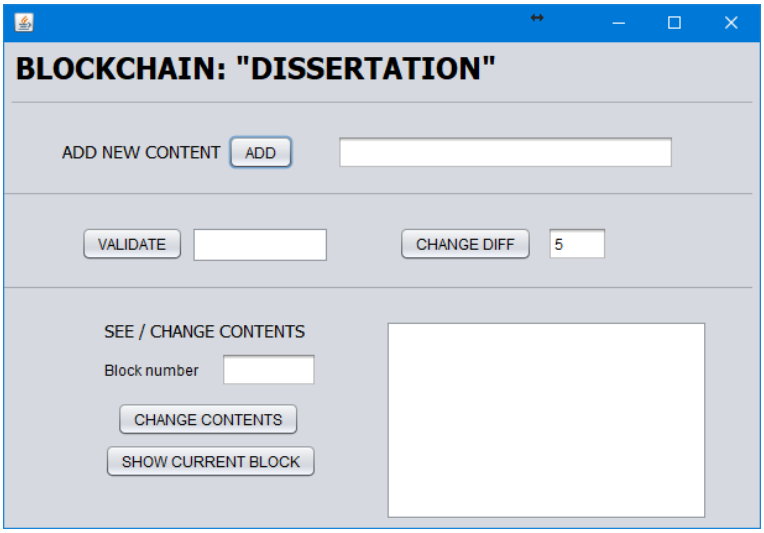

Figure 3.3: Management window.

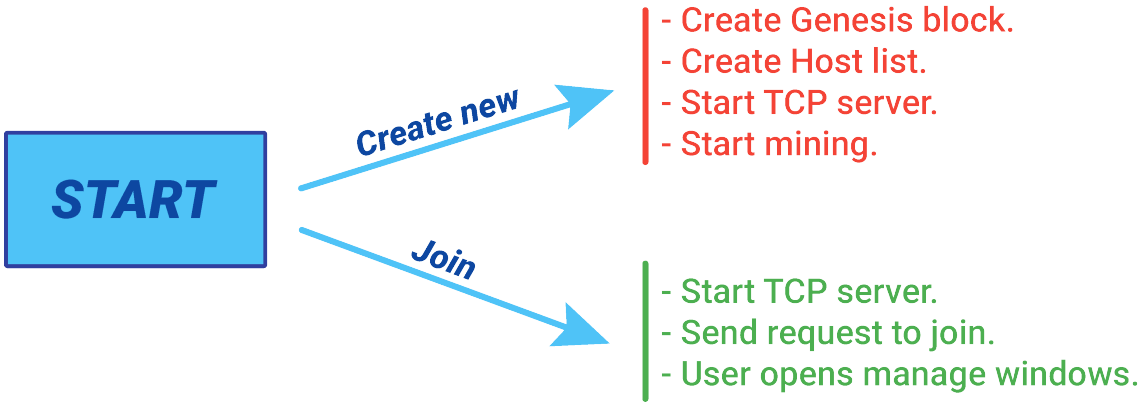

Figure 3.4: Start diagram.

3.2 Second iteration: dynamic difficulty

On this second iteration, we use the code from the first one and add the capacity to dynamically change the difficulty depending on the mining rate. The code can be found in:

https://github.com/pepebndc/Blockchain-JAVA/tree/v2-dynamic-diff

As a design key, we want to have a block every minute (more or less). This makes the blockchain fast enough that we can easily and quickly check new contents, while avoiding a flooding in blocks (making a lot).

Difficulty will be dynamically changed each time a node finds a valid hash for a block with an index multiple of 10 (10, 20, 30...). This avoids everyone to be checking the difficulty continuously and possible missunderstandings between the nodes.

In order to do this in a reliable way, we must adapt first some of the code we previously created. In each block, the current time will be saved in milliseconds from epoch using the *System.currentTimeMillis();* Java method.

We choose to check with the previous 10 blocks to avoid changes when a single block is mined too fast or too slow, and if on average, a block took $\pm$ 15s to be mined, the difficulty is adjusted by 1. This way, if a block takes on average 30 s, the difficulty is increased and the average mining time will be increased.

```
 //CHECK FOR DIFFICULTY CHANGES NEEDED IF BLOCK IS MULTIPLE OF 10
if (length % 10 == 0) {
    long thisTime = Long.valueOf(mined[2]);
    long prevBlockTime = main.TFG.getChain().get((length - 10)).getTime();
    long difference = thisTime - prevBlockTime;
    int differenceInSeconds = (int) difference / 1000;
    System.out.println("difference in seconds: " + differenceInSeconds);

    int newDiff = main.TFG.getDiff();;
    if (differenceInSeconds > 750) {
        newDiff = main.TFG.getDiff() - 1;
    }

    if (differenceInSeconds < 450) {
        newDiff = main.TFG.getDiff() + 1;
    }
    //set the new diff
    System.out.println("new diff after comprobation: " + newDiff);
    main.TFG.setDiff(newDiff);

    //notify the network about the new difficulty
    Iterator<String> it = main.TFG.getHosts().iterator();
    while (it.hasNext()) {
        String host = it.next();
        try {
            if (!host.equals(InetAddress.getLocalHost().getHostAddress())) {
                TCPclient.sendNewDiff(host);
            }
        } catch (UnknownHostException ex) {
            Logger.getLogger(manage.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
}//end of difficulty changes
```

We also need to adapt the GUI in order to let the user see the current difficulty. For now, the user will still be able to manually change the difficulty of all the network, but will not be able to stop the dynamic changes (when the blockchain gets to a block multiple of 10, the difficulty will adjust). The result will be this:

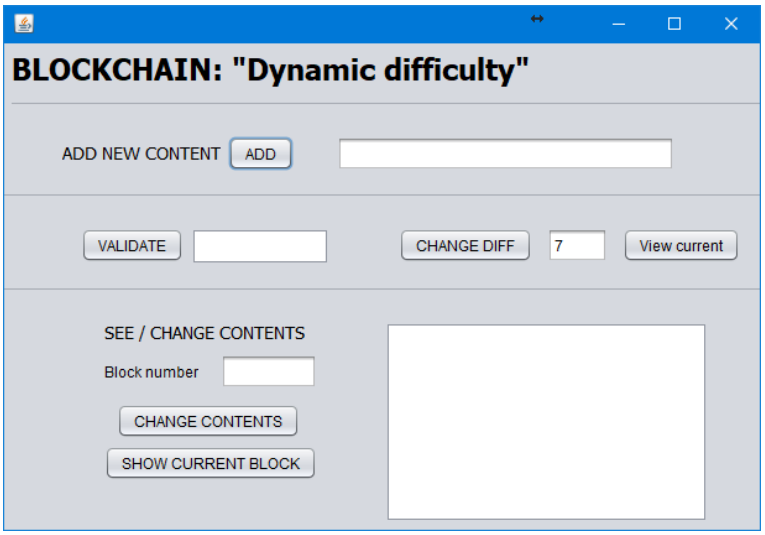

Figure 3.5: Management window with dynamic difficulty

3.3 Third iteration: SHA-512 hash

Currently, the hashing algorithm we are using is MD5. This algorithm is known for being insecure and easily breakable, but it was easy to implement for the early stages of development. Now, before we start implementing more security on the blockchain, we are going to change to SHA-512. Without getting into much detail on the specifics of this hashing algorithm, this technology is considered secure and even "overkill" on most of the implementations. Even Bitcoin uses a less secure hashing algorithm (SHA-256), so our blockchain will be properly secured.

The only change in the code is in the *computeHash* method and it can be found in:

https://github.com/pepebndc/Blockchain-JAVA/tree/v3-sha512-hash

```
static String computeHash(int index, long time, String data, String lastHash,
    int nonceInt) throws NoSuchAlgorithmException, UnsupportedEncodingException {

    String plaintext="";
    String hashtext="";
    plaintext = index + time + data + lastHash + Integer.toString(nonceInt);

    try {
        MessageDigest md = MessageDigest.getInstance("SHA-512");
        byte[] bytes = md.digest(plaintext.getBytes("UTF-8"));
        StringBuilder sb = new StringBuilder();
        for(int i=0; i< bytes.length ;i++){
        sb.append(Integer.toString((bytes[i] & 0xff) + 0x100, 16).substring(1));
    }
    hashtext = sb.toString();
    }
   catch (NoSuchAlgorithmException e){
    e.printStackTrace();
   }

    return hashtext;
}
```

This will create a 128-character hash for each block. Computing SHA-256 hashes is more computationally demanding than MD5, so we change our concept of difficulty. Now, instead of the number of '0's at the beginning of the hash, we will introduce intermediate steps in order to make more viable achieving the 1-minute-per-block goal.

Each character on a SHA-512 hash can take any hexadecimal value (0-9 and A-F), for a total of 16 different characters.

This security is needed in order to prevent possible attacks where the attacker knows the hash (it is public) and reverse-engineers the contents of the block and fakes those contents, as we explained in the blockchain basics chapter (2.3.2).

3.4 Fourth iteration: users

Contents on a Blockchain are immutable, but identifying who created a certain transaction or added specific information is crucial to the correct operation of the network. On the next two iterations we will approach this issue and modify the project accordingly. On this fourth iteration, we will be adding basic users and identifying those users through the network, while on the fifth iteration we will redesign how data is stored in blocks and how users "sign" those blocks. As usual, all the code corresponding to this iteration can be found in:

https://github.com/pepebndc/Blockchain-JAVA/tree/v4-Users

But first, let's start with the users. In a blockchain network, users are usually identified by an unique address inside the network. In our network, an address is a 50 character long String with randomly generated characters in it, including lowercase and capital letters, as well as numbers. With a 50-character-long random address, there are in total $62^{50} = 4.16 * 10^{89}$ different addresses, effectively making it impossible for two users to generate the same address. Addresses can't be created sequentially since this will create two problems:

1. The first issue is a security issue. If addresses are created sequentially, you will certainly know that the previous address to you does exist, and every combination previous to you is linked to an existing user, therefore opening the gates for a possible attack.

   With random addresses, sequential attacks can't be performed, although anyone connected to the network knows the address of all the users.

2. The second problem is a logistic problem. As we will see later on the section, users are created before connecting to the network, and you can't know what addresses are being used and which onces aren't.

On this design, we will use asymmetric cryptography (RSA 1024) to identify and validate users on the network. Asymmetric cryptography systems work providing a public and a private key to each user. The public key is supposed to be known by everyone while private key must remain secret. This mechanism allows the implantation of many things, but we will be focused on two of them:

1. When you encrypt a content with your Private key, anyone can decrypt it using your publicly available Public key. This allows anyone to certify that the content was sent by you and only you, since no one else has access to your Private key.

2. On the other hand, when someone encrypts a content with your Public key, you and only you can decrypt it and know the original contents. This is used to privately send data that only the designated receiver can read.

Once we have some base knowledge, we can further explain how we have implemented it on our project. We have created two new objects: `User` and `LocalUser`. `LocalUser` contains the *Name*, *Address*, *Public Key* and *Private Key* of a user, and also implements a method to create a new `LocalUser`.

```
public static LocalUser create(String newName) throws NoSuchAlgorithmException {

    String newAddress = "";
    KeyPairGenerator keyGen;
    KeyPair pair;
    PrivateKey newPrivate;
    PublicKey newPublic;

    //generate address
    char[] chars =
        "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789"
        .toCharArray();
    StringBuilder sb = new StringBuilder();
    Random random = new Random();
    for (int i = 0; i < 50; i++) {
        char c = chars[random.nextInt(chars.length)];
        sb.append(c);
    }
    newAddress = sb.toString();

    //generate keys
    keyGen = KeyPairGenerator.getInstance("RSA");
    keyGen.initialize(1024);

    pair = keyGen.generateKeyPair();
    newPrivate = pair.getPrivate();
    newPublic = pair.getPublic();

    return new LocalUser(newName, newAddress, newPublic, newPrivate);
}
```

`User` is an object that contains the same information as `LocalUser` excluding the private Key. This object is designed in order to store through the network the information of each user.

Once the users are created, we have to modify some of the operations on the blockchain and the management console.

Joining the network

Now that users can be identified, we have to make sure they are who they say they are when joining the network. This mainly means corroborating that their public and private keys match. In order to do this, we have to modify the connection process.

Previously, users simply had to send a request to join, and they would be accepted. Now, users are sent a "challenge" they have to solve and prove their keys are in order.

This challenge is simply a random string encrypted with the public key of the new user and sent in bytes in order to be decrypted by the user. Once the decryption happens, the user sends the solution and if it matches the original challenge, the user is admitted in the network.

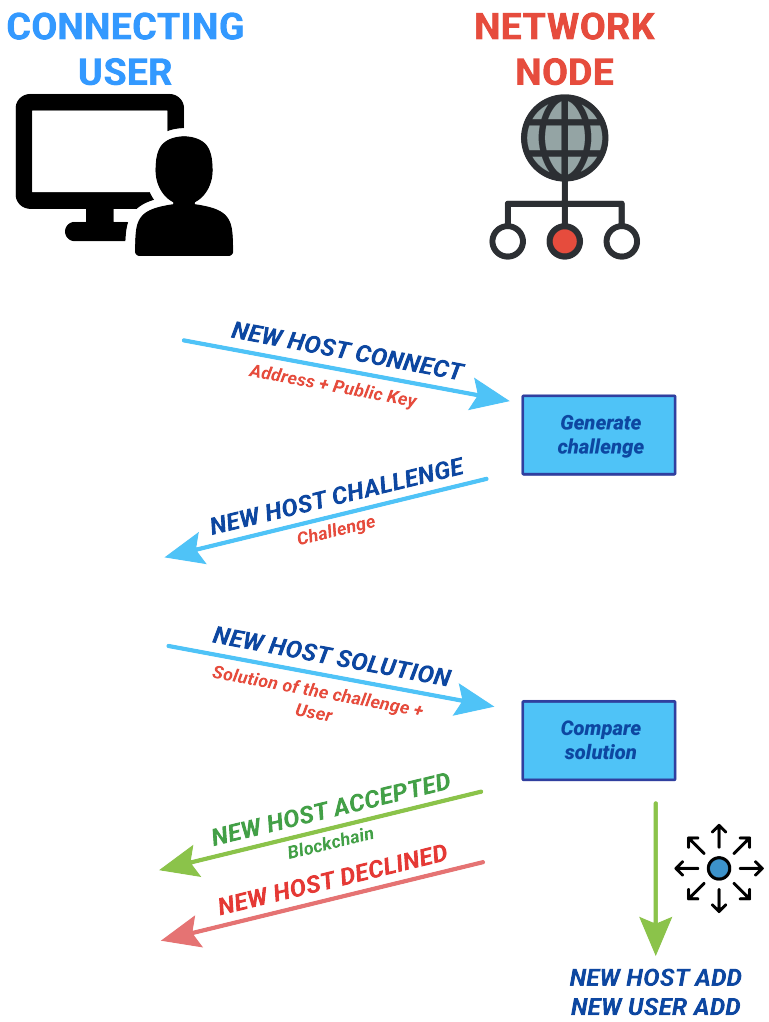

Figure 3.6: Connection sequence with users.

```
//encrypt the challenge with the public key of the new host
PublicKey pubKey = command.getPubKey();
Cipher encrypt = Cipher.getInstance("RSA/ECB/PKCS1Padding");
encrypt.init(Cipher.ENCRYPT_MODE, pubKey);
byte[] encryptedMessage = encrypt.doFinal(plainTextChallenge.getBytes());
System.out.println("I have created a challenge, encrypted: " + encryptedMessage);
TCPclient.sendChallenge(ClientIP, encryptedMessage);
```

This process works the same way either if it is a new user or an existing one who is trying to connect. Existing users have to load their credentials before connecting (explained on the next subsection).

Since now there are two different categories in the network (users and hosts), we have to check if those are already on the network before notifying the rest of the hosts. A host is a node of the network and is identified by its IP address. Hosts are removed from the list when they disconnect, while users are saved forever.

Managing users

When using a service, users expect to be able to log in from different hosts or locations, as well as going "offline" and then connecting again. When using a password-based service, the user simply has to remember the password and they will be able to log in from anywhere. On this asymmetric credential-based system, users must save their credentials (name, address, public and private key) somewhere safe, since it's almost impossible for anyone to remember all those data.

In order to solve this issue, we can save and load credentials from a file. This file contains a `LocalUser` object and has an extension *.BCpepe* and can be loaded from our project.

Minor changes

We have also changed the way data is inserted in the chain. Now it's a String with the address of the sender and the data, instead of just the data. This is a minor step and a guide to the next iteration of the project, where transactions are completely redesigned.

Redesigned windows

Implementing users force some changes in the windows of the program.

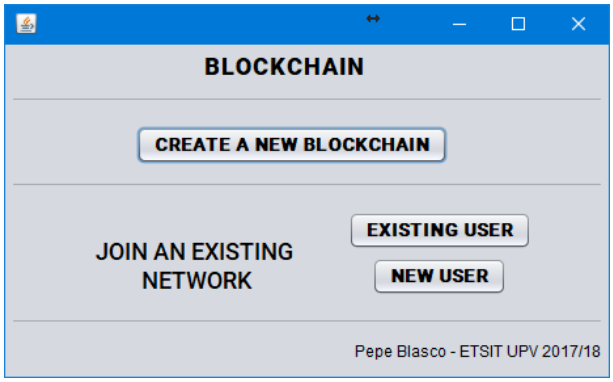

Figure 3.7: Start window.

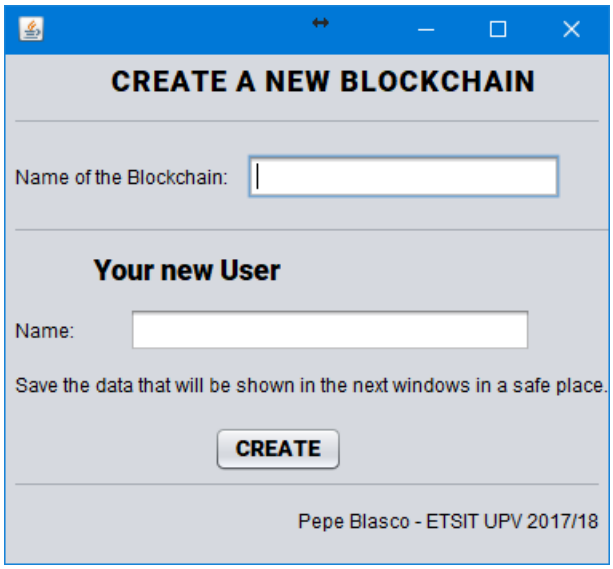

Figure 3.8: Creating a new Blockchain.

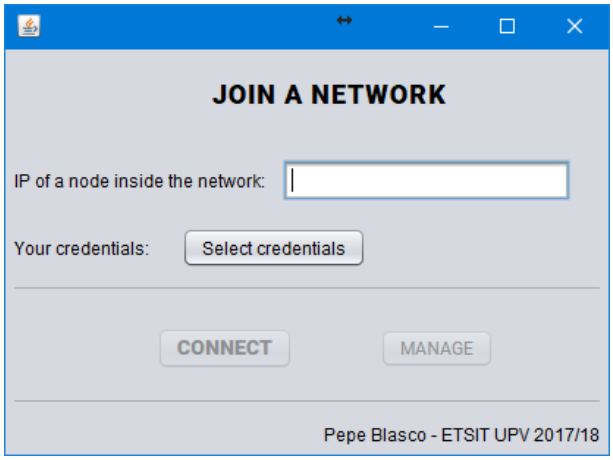

Figure 3.9: Joining the network with an existing user.

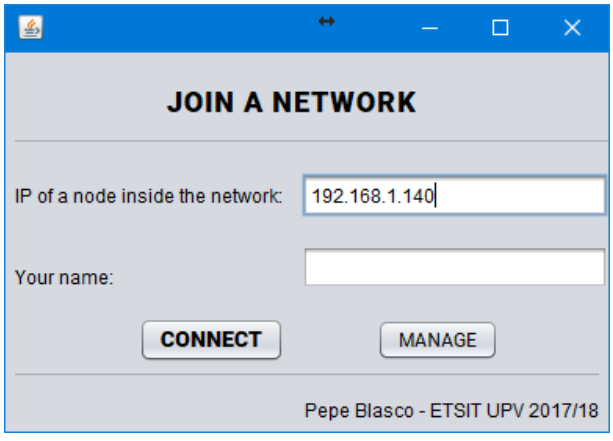

Figure 3.10: Joining the network with a new user.

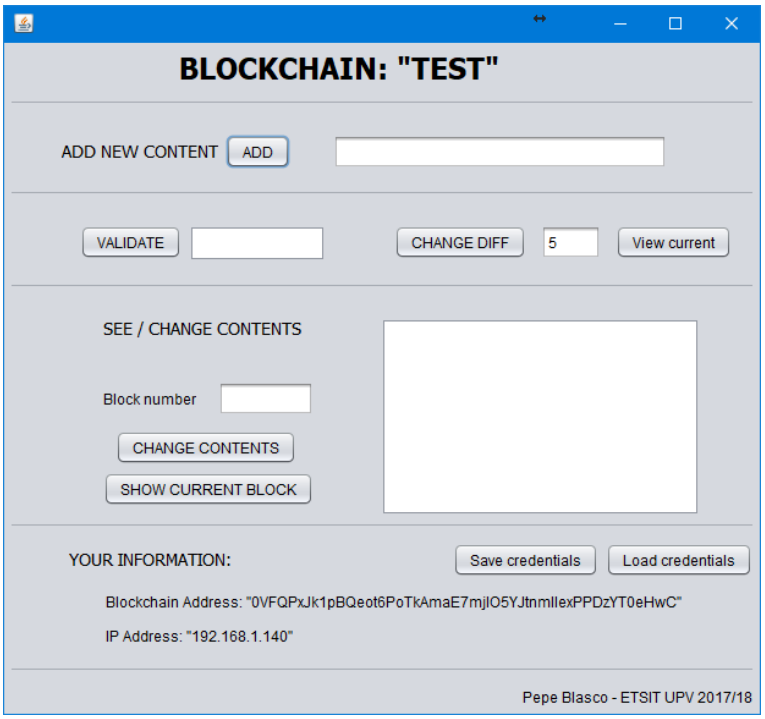

Figure 3.11: Management window with save/load options for users.

3.5 Fifth iteration: transactions

In this iteration, we are changing the basic design of our blockchain network. We used to add simple strings to the blockchain as contents and now we are going to include transactions. A transaction is an interaction of a user with the blockchain and it includes the contents the user wants to include (which are encrypted, more on that later), as well as the date it was created at and the users who interact with it. The code for this iteration can be found in:

https://github.com/pepebndc/Blockchain-JAVA/tree/v5-Transactions

There are two essential types of transaction, those which are saved into the blockchain by only one user, and those agreed by two or more users.

Before we take a deep look on transactions, in this iteration we have added an option to start / stop mining, since some users might want to control the CPU use of their computer. This is possible due to an introduction of a "kill switch" on the mining thread.

Now, focusing on transactions, we have to maintain the idea that the blockchain is a concatenation of blocks, and each block stores information sent by the users to the network. With this redesign, that information is now a list of transactions created by users.

It is important to be aware that one of the key functions of a blockchain is security. Security is implemented in different ways, and in this blockchain several techniques combine to create one of the most (if not the most) secure method of storing data. In our blockchain, security takes place when hashing the blocks with SHA-512 and connecting the blocks between them so that they can not be edited without invalidating the whole chain.

When we created the users, we went one step forward and forgot about traditional passwords and implemented an RSA-based system with public and private keys. Those keys are used to certify that the user is who they say they are, avoiding identity thefts.

Now, with transactions, we take a giant leap forward ensuring that the content that is included by the users is stored securely and every user on the network can certify who created that transaction.

Each transaction is identified by a 75-character-long String and the contents of the transactions are secured by an encryption process combining RSA and SHA. RSA, as we explained in the previous section, is an asymmetric encryption algorithm where each user has a private and public key. On the other hand, SHA allows for a secure hash of contents. Both of these processes are secure on their own, but combined are almost impossible to hack while being able to create a beautiful and optimized encryption process.

Depending on the type of transaction it is (sent to the network or to other users), the process differs a bit. When a user wants to send a transaction directly to the network, the user hashes the content of the transaction, and then encrypts it with his private RSA key. This way, anyone with the public RSA key would be able to decrypt the hash and compare with the hash of the content he received and verify its integrity

If the transaction is created by two or more users (agreed transaction), the first user (creator) performs the same process as before but instead of sending the transaction into the network for mining, it is stored in a pending-transactions list. Then, the second user (receiver) can see this pending transaction and read the contents. He hashes the contents and compare it to the signed hash from the creator, if he agrees to it then he signs the hash with his private RSA key.

```
        //find the number of semicolons
        int count = receivingAddresses.length() -
            receivingAddresses.replace(";", "").length();
        //separate the string
        String[] address = receivingAddresses.split(";");

        //hash contents of the field
        String myHashedContents = main.findHash(contents);

        //Sign the hash with my private key
        byte[] mySignature = Transaction.encryptSHA(myHashedContents.getBytes
            (Charset.forName("UTF-8")), null,
            main.getLocalUser().getPrivateKey());

        //create the random string for the ID
        char[] chars =
            "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789"
            .toCharArray();
        StringBuilder sb = new StringBuilder();
        Random random = new Random();
        for (int i = 0; i < 75; i++) {
            char c = chars[random.nextInt(chars.length)];
            sb.append(c);
        }
        String transactionID = sb.toString();

        //create the transaction
        //1.create the list of users and signatures and add yourself
        List<String> newUserList = new ArrayList<>();
        newUserList.add(main.getLocalUser().getAddress());

        List<byte[]> newSignatureList = new ArrayList<>();
        newSignatureList.add(mySignature);

        //add the receiving addresses
        for(int a=0; a<1+count; a++){

            newUserList.add(address[a]);
            newSignatureList.add(null);

        }

        //2. create the transaction
        Transaction t = new Transaction(transactionID, newUserList, contents,
            newSignatureList, System.currentTimeMillis(), 1);
```

In this scenario, if someone wants to read the contents of a transaction and verify that everyone involve agreed, he hashes the contents and compares them to the signed hash of the creator, checking that the content he is seeing is what the creator and the rest of users agreed on.

The graphic interface has changed and adapted to this new structure. Now, when users search for the contents of a block, they get a list of transactions that were included in the block. Then, they can search for the contents of a specific transaction in a new designated area.

Every user can see all the transactions on the network once they have been signed by every user. Transactions that are pending to verify will only be visible on the user interface

of those who are part of it. This version will be the base for future development and will be used for implementing a proof of existence platform.

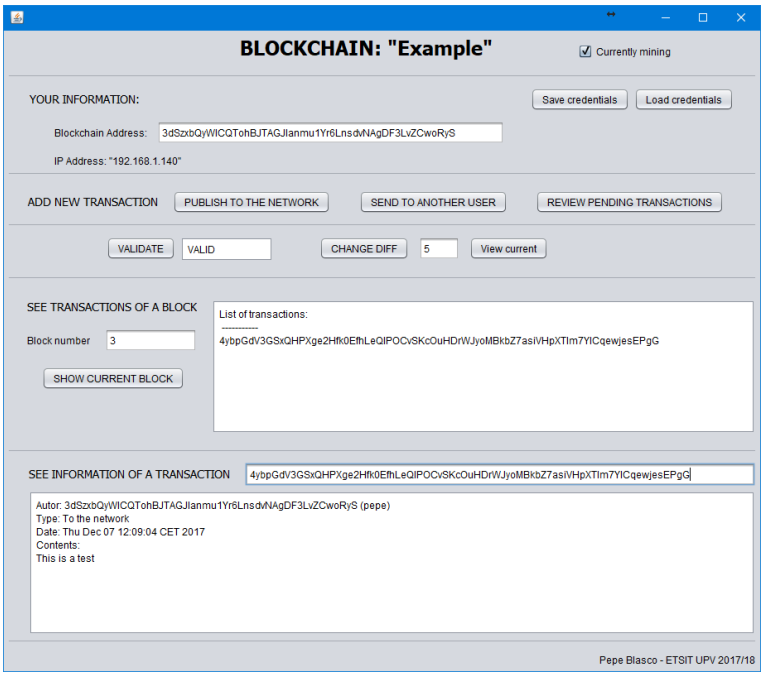

Figure 3.12: Updated management window.

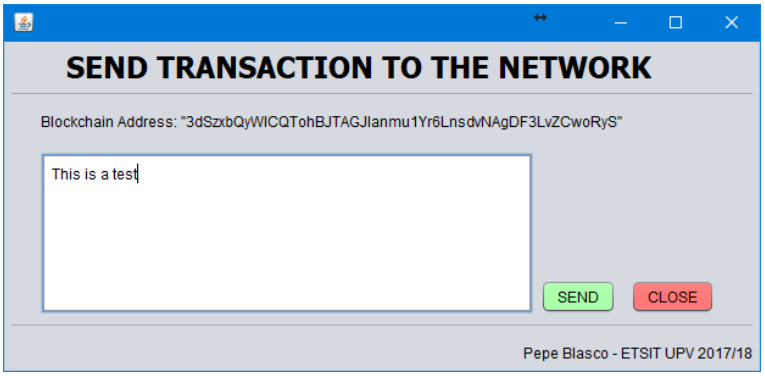

Figure 3.13: Send transaction to network.

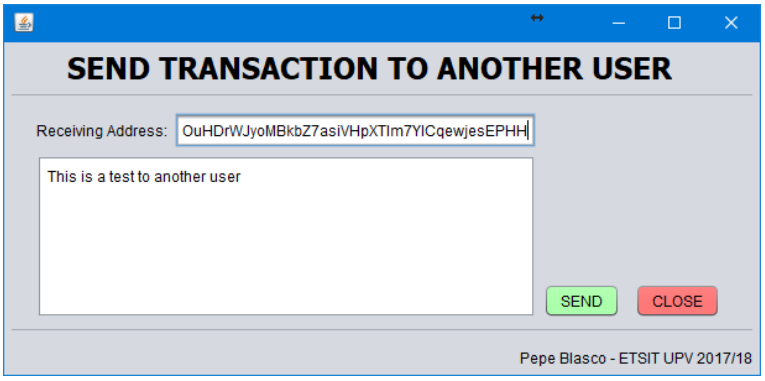

Figure 3.14: Send transaction to another user.

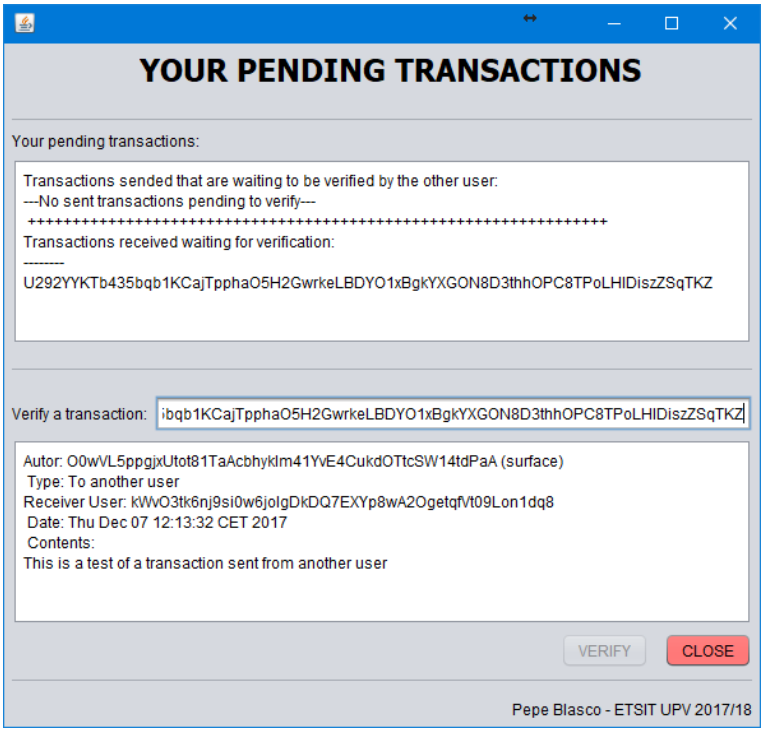

Figure 3.15: Control and verify pending transactions.

CHAPTER 4

Development of a Proof of Existence platform

4.1 Introduction to PoE

Proof of Existence[8], or PoE for short, is the digital equivalent of sealing an envelope and securing it in a trustful place.

However, doing this in a traditional way is very complicated and involves a lot of risks. For instance, you have to completely trust a single point of failure, such as a notary or a bank. Also, once it has been tampered, you have no information on who, when and where interacted with this document.

Digital computers revolutionized this by storing a secure copy of the hash of a document and checking whether or not the hash was the same when you wanted to verify this document, but you still had the issue of the centred point of failure.

However, in blockchain, as we saw in the introduction of this project, there is no single point of failure, since the information is safely stored on every node in the network, therefore having redundancy and not a single point of failure.

Proving the existence of data at a certain point in time can be very useful for many people, such as entrepeneurs, universities and even attorneys. Timestamping data in an unalterable state while maintaining confidentiality is perfect for legal applications. Attorneys (or clients) can use it to prove the existence of many documents including a will, deed, power of attorney, health care directive and so on without disclosing the contents of the document. A person can use the blockchain to prove that a document, such as a will, that will be presented in court in the future is the same unaltered document that was presented to the blockchain at a prior point in time.

Other applications and uses for PoE are timestamping a document to prove that the document existed in a certain point in time (for example, for bachelor's thesis and master's thesis).

It is also possible to use this kind of platforms to perform fair public contest between companies that apply for a certain project. Currently, if company A turns in their offer and the public register is corrupted, company B might get their hands on A's offer, therefore modifying theirs and having a better deal. This can be avoided by registering the documents in a blockchain-based PoE platform that stores the hash and the timestamp in a secure way, and once after the deadline for the project is met, companies would present the full offer and the verification that the offer matches the one registered on the blockchain.

4.1.1. How it works

Storing data in a blockchain is expensive, so we need to find a way to reduce the number of bytes stored in order to make things go faster and cheaper while maintaining the security and uniqueness we need.

This is why we use, once again, hashes. Hashing a document creates a unique hash that works as an "ID" to that document. If the document is changed in any way, the hash will be different, therefore allowing us to detect modifications of a registered document.

The process of registering a document into a PoE platform consists in uploading the document to the platform, where the hash will be computed and stored into the blockchain. Then, when you want to verify the integrity of the document, you have to recompute the hash and compare it with the one which was initially stored. If the document was modified in any way, the hash will not be the same.

It is worth mentioning that some file formats change over time, for instance a Word document is different every time you open it, therefore the verification would always be

wrong. This is why it's highly recommended to verify persistent documents such as PDF, images or plain text notes.

4.2 Development

In order to create our own Proof of Existence platform, we will adapt our previous work to match the requirements we need. All the code can be found in:

https://github.com/pepebndc/Blockchain-JAVA/tree/v6-PoE

First of all, we need to plan how we will transfer and register the hashes. After analyzing our previous work, it is trivial that by simply introducing the hash into the "contents" field in our transaction object, the network will behave correctly and all the requirements will be matched.

On the other hand, we need to adapt our user interface to allow users to upload files and send the hash to other users, while also allowing the receiving user to check if the document they are verifying matches the one they own.

In order to do this, the main interface has been adapted to correctly show all the options regarding the documents.

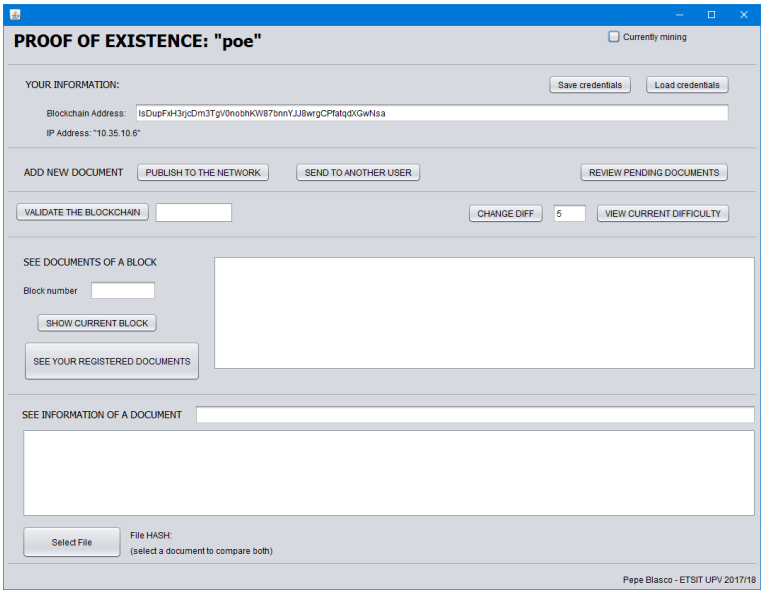

Figure 4.1: Main window of the PoE platform

4.2.1. Registration of documents

When adding a new document, the user is presented with a user-friendly menu where they can upload a file from their computer.

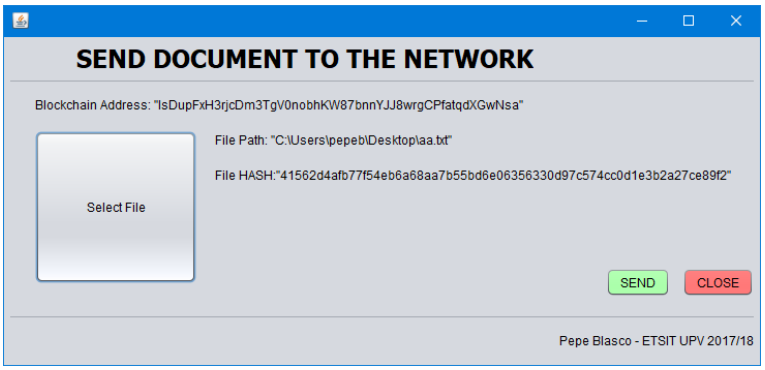

Figure 4.2: Sending a document to the network

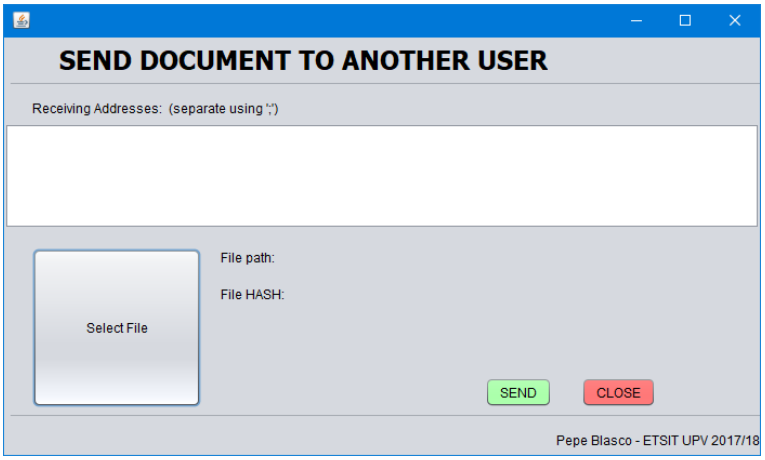

Figure 4.3: Sending a document to another user

4.2.2. Verification of documents

As we can see, the user has now all the options to send, publish and verify documents in the network, with the addition of the option to compare a registered document with one of their own at the bottom of the window.

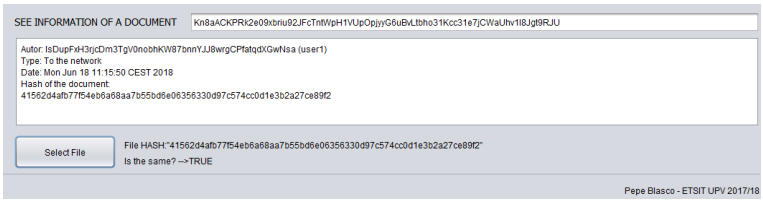

Figure 4.4: Verification of the same document

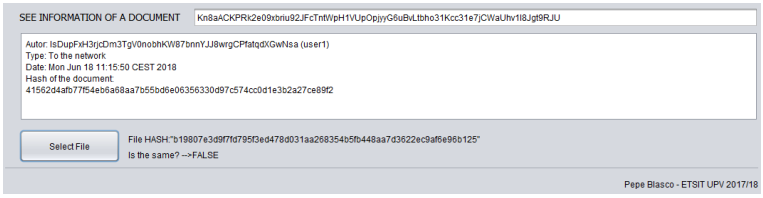

Figure 4.5: Verification of an altered document

When a second user needs to verify a document that a first user has sent, he is also presented the option to verify the document before signing the transaction.

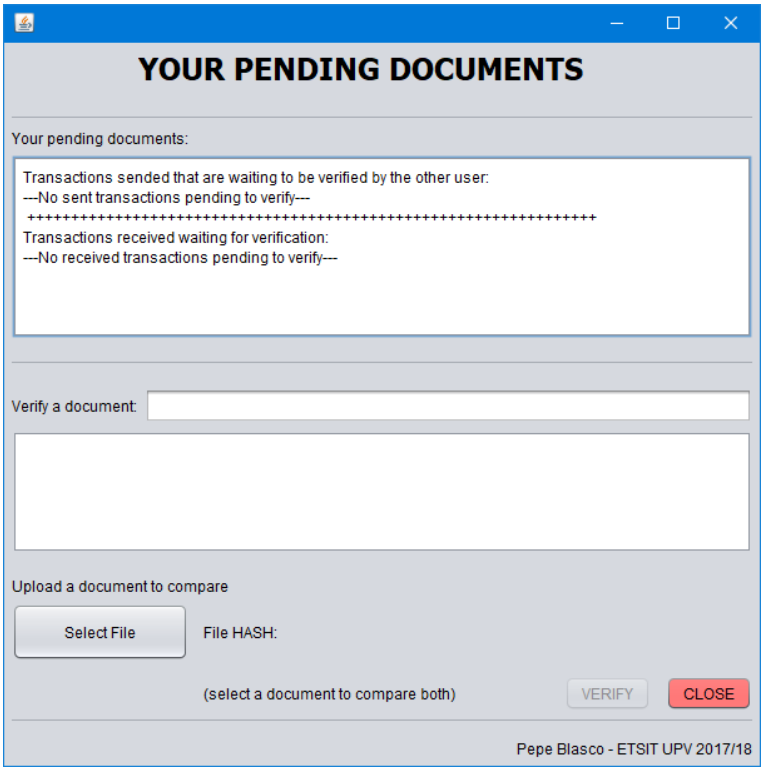

Figure 4.6: Verification of a document before signing

4.2.3. List of registered documents

The user needs a way to list all of their documents. This is why it is implemented a method to export to a plain text file all the documents he has registered. The exported file it is includes the transaction hash, the document hash, the block in which the transaction was included and the type of transaction it was.

With this information, the user can go to the main interface of the platform and look for the transaction hash they are interested in, which will give them more information including a timestamp and other user's info.

```
---------- LIST OF DOCUMENTS SAVED BY ----------
lsDupFxH3rjcDm3TgV0nobhKW87bnnYJJ8wrgCPfatqdXGwNsa
------ DATE: Mon Jun 18 11:36:24 CEST 2018 ------

LIST:

-------------------------------------------------
Transaction in block 2
Transaction hash: Kn8aACKPRk2e09xbriu92JFcTntWpH1VUpOpjyyG6uBvLtbho31Kcc31e7jCWaUhv1l8Jgt9RJU
Document hash: 41562d4afb77f54eb6a68aa7b55bd6e06356330d97c574cc0d1e3b2a27ce89f2
Type:To the network.
-------------------------------------------------

Transaction in block 29
Transaction hash: NeA62ST9u4yGvbBxVArz11AU5aNHDqTXlJYVNjLCf9CkQ0j6Lfv2MAEO7QzOXHmwiV4i92mIxH8
Document hash: d4fa09bd70f70c5eea28f46001c161975bd9775dcd85aad3fd5b5fb2e82e79c9
Type:To the network.
-------------------------------------------------

Transaction in block 29
Transaction hash: BSMjyvfkbFVmbrKvB6cpMLoNm0psBtD2WOnFHqdK8IY8aa03A2h704SB04YCf7iVzqd1DvU5pIh
Document hash: 3211ad849a7480ddaefa419c8742bb1fdee9f689216fc097f5f036d6d0c4c91f
Type:To the network.
-------------------------------------------------

Transaction in block 29
Transaction hash: OCmbZEodAoHgtdgu9nQsHPepmVAcQ1d8SitXwvu9mXdS0kamp80HxT6ltbVhBtlADbDfWcPuQeb
Document hash: 388363d36852582a32d3e0e79f7e3f0a091ceca2cb8a0e4496cd47e952c51a3b
Type:To the network.
-------------------------------------------------

------------ END OF THE LIST -----------------------

```

Figure 4.7: List of users' transactions

CHAPTER 5

Other blockchain applications

As we have seen, blockchain is not only able to transform the technological part of business, but it also transforms the business part of business. With blockchain technology new paradigms and opportunities are arising across industries. [9][10][11][12]

In this chapter, we will collect different use cases that are being redefined by blockchain so that we can get a real sense of the broad range of opportunities this technology offers.

From money to law, including smart cities and car rental, disrupting digital identity and audits, blockchain technology has proven to be useful in many more contexts than what we could initially imagine. As we analyse on the beginning of this document, the key points of blockchain technology are:

- **Decentralisation**: Data is stored on different nodes and information does not depend of a single central identity.
- **Trustless**: Consensus mechanisms are applied, therefore trusting the network code rather than a single set of identities.
- **Persistent**: Transactions are inmutable and timestamped by concatenating hashes and invalidating any modification on the network.

With this three base points, we are now able to identify and analyse some of the most common use cases for this technology. This analysis will be rather short and concise since we are not implementing them, just identifying them for future development of this technology.

5.1 Money

Money is, and probably will always be, the key gear on our society. Although money is now centralised on banks, it has not always been that way. People used to trade directly among them with the goods they had, society used to be a (very rudimentary) P2P network that shifted to central banks for many reasons.

However, money is now returning to a P2P model. Blockchain has proven to be a secure, fast and reliable enough technology to send and receive money. Currently, a big part of blockchain implementations are related to money.

Projects like Bitcoin, Litecoin, Monero and many more allow their users to transfer value among them in a decentralised way, allowing anyone to participate and validate every move performed on the network, therefore avoiding the complications and administrative delays and issues central banks could introduce.

Advantages:

- **Secure**. Transactions are secure and will never be erased, assuring that once you receive some money, it will not magically disappear.
- **Trustless**. Users avoid the need to trust an external organisation such as a bank, instead they can actively participate on the network.

Disadvantages:

- **Throughput**. Current implementations of blockchain technology offer a low volume of transactions per second, making it difficult to implement a global network with the required volume of millions per second.

Examples:

- https://bitcoin.org/
- https://litecoin.org

5.2 Tokenization

On our lives, everything is a token we can change for another token (in most of the cases, a token named "euro"). For instance, we change a token of "banana" for a token of "euro".

So, what if we digitalize those tokens and allow users to buy, sell and trade them? There are many standars on tokenized assets (ERC20, ERC 721...) that allow platforms to tokenize almost everything. You have a house, you can tokenize it. You have a kilogram of paper, you can tokenize it. You have "one hour worth of energy", you can tokenize it.

Virtually everything is tokenizable, therefore allowing for an easy trade and verification of authenticity, owneability and usage.

You could buy a token of "one hour of driving a Tesla car" when you go to another city and need to rent a car for only one hour. The system will know when and where you picked up the car and when and where you left it, the recharge cost and maintenance fee will be calculated and the value of the token "one hour of driving a Tesla car" will vary accordingly.

Examples:

- https://www.cryptokitties.co/
- https://crypto20.com/en/

5.3 Supply chain

When you go to a sushi restaurant, you trust the restaurant to buy and preserve good quality fish so that you don't get any disease. The same thing happens when you buy a diamond and want to know where it came from.

Blockchain allows companies to create a global, trustless network where everything is tracked and verified. When you get to the sushi restaurant, you could get a QR code next to your nigiri identifying when the fish was caught, the temperature of the ship while transporting and whether or not the cold chain was preserved.

5.4 Content creation monetization

Some of the biggest problems on the content creation industry is transparency, royalty distribution and ownership rights. With blockchain and smart contracts anyone could create a decentralized, comprehensive and accurate database of music rights.

Using that network, I could play a song on my Youtube video and part of my benefits would go directly to the artist. That artist could also have setup an smart contract that divides the benefits of the group among the members of the group.

Examples:

- https://musicoin.org/
- https://opus.audio/

5.5 Voting and governance

In many of the countries in the world, there are democracies as political systems. In those systems, citizens cast a vote to decide on who should rule the country and what rules should be implemented.

However, this is not a very effective method as it is currently being done (at least in Spain), since we can only express our opinion once every four years and we can't change our opinion in the meantime. Wouldn't it be great if we could create an Smart-Contract which could verify the identity of everyone on Spain, allowing them to vote not only every 4 years, but maybe even every day for each separate proposed rule?

This way, everything would be truly democratic and we could cut a huge part of the cost of democracy (salaries and a huge waste of paper every election). This might sound like a crazy idea that is far away in time, but it has been already implemented in places like Sierra Leone [13] and many cities and companies are working on projects to make this idea a reality.

CHAPTER 6
Conclusions

In this project we have seen how Blockchain technology is a reality and is disrupting some of the biggest parts of how we currently see the world. After completing this dissertation both the student and the reader have a broad understanding of what Blockchain is and how it works.

We have developed our own Blockchain using Java and the knowledge acquired during the degree regarding cryptography and networking. We structured the project in a clear and organized way that allowed us to develop step by step a complete blockchain and modify it for a demonstration of a very interesting use case as is the verification of documents.

The implementation of our blockchain is completed, but could be improved in a further revision by increasing the security of the P2P network and adding other methods to interact with the network, such as a server or an API.

At the end of this project, the biggest thing we can conclude is that Blockchain is not only money and that the potential of this technology of future developments will disrupt and improve our lives, and I hope to be part of that revolution.

Bibliography

[1] 4 characteristics that set blockchain apart. https://ibm.com.

[2] Genesis block. https://en.bitcoin.it/wiki/Genesis_block

[3] Block. https://en.bitcoin.it/wiki/Block.

[4] The difference between PoW, PoS and PoI algorithms. https://steemit.com.

[5] NEM Technical Reference Introduces Reputation-Enhanced 'Proof of Importance'. https://cointelegraph.com.

[6] The Bitcoin Mining Arms Race: GHash.io and the 51% Issue. https://www.coindesk.com.

[7] A Short Guide to Bitcoin Forks. https://www.coindesk.com.

[8] Proof of Existence. https://www.newsbtc.com/proof-of-existence/

[9] 17 Blockchain applications that are transforming society. https://blockgeeks.com/guides/blockchain-applications/

[10] 8 Blockchain Applications That Could Help Your Small Business. https://www.upwork.com/hiring/for-clients/8-blockchain-applications-help-small-business/

[11] What Are the Applications and Use Cases of Blockchains? https://www.coindesk.com/information/applications-use-cases-blockchains/

[12] Popular Use Cases of Blockchain Technology You Need to Know https://hackernoon.com/popular-use-cases-of-blockchain-technology-you-need-to-know-df4e1905d37

[13] https://techcrunch.com/2018/03/14/sierra-leone-just-ran-the-first-blockchain-based-election/?guccounter=1 https://techcrunch.com/2018/03/14/sierra-leone-just-ran-the-first-blockchain-based-election/?guccounter=1