



INSTITUTO
UNIVERSITÁRIO
DE LISBOA

Blockchain Technology for the Construction Industry

Tiago Furtado Piques Martins Mota

Master in Open Source Software

Supervisors:

PhD Mauricio Breternitz Jr., Principal Researcher,
ISCTE - IUL

PhD Ricardo Pontes Resende, Assistant Professor,
ISCTE - IUL

September 2020

Acknowledgments

I would like to thank Professor Mauricio Breternitz Jr. and Professor Ricardo Pontes Resende for their support and availability throughout the development of this thesis. Their revisions and suggestions were very important to improve the quality of this document.

I would also like to thank all my course colleagues for the moments we had and to all my friends for their motivation.

Finally, I thank all my family for their constant motivation in the fulfilment of this course, especially Marta Cabral for her constant companionship and support.

Resumo

Um dos desafios que a indústria da construção enfrenta é a falta de confiança entre os intervenientes e os sistemas de partilha de informação. Blockchain é uma tecnologia disruptiva e emergente que pode ser usada para adicionar imutabilidade, confiança e transparência à informação. A presente dissertação propõe uma plataforma que pretende mitigar o problema de partilha de informação na indústria da construção utilizando a tecnologia blockchain. A plataforma permite manter um registo imutável das alterações efetuadas em ficheiros partilhados entre os vários intervenientes da obra e simular assinaturas de documentos que possam ser, posteriormente, verificadas. Foi desenvolvida uma prova de conceito utilizando a rede Ethereum sendo, de seguida, utilizada para avaliar a influência do preço unitário do *gas* na duração de execução da transação e o seu custo. Conclui-se que a tecnologia blockchain pode auxiliar a partilha de informação na indústria da construção.

Palavras-Chave: Blockchain; Indústria da construção; Smart contract; Ethereum.

Abstract

One of the challenges that the construction industry faces is the lack of trust between participants and information sharing processes. Blockchain is a disruptive and emerging technology that can be used to add immutability, trust and transparency to information. This dissertation proposes a platform that aims to mitigate the problem of information sharing in the construction industry using blockchain technology. The platform allows to keep an immutable record of file interactions between construction participants and simulate document signatures that can later be verified. A proof-of-concept was developed using the Ethereum network, which was also used to evaluate the *gas* price influence in the execution duration of the transaction and its cost. It is concluded that blockchain technology can support information sharing in the construction industry.

Keywords: Blockchain; Construction industry; Smart contract; Ethereum.

Table of Contents

Acknowledgments	iii
Resumo.....	v
Abstract.....	vii
Table of Contents	ix
List of Tables	xiii
List of Figures.....	xv
Abbreviations	xvii
Chapter 1 - Introduction	1
1.1 Scope	1
1.2 Motivation	1
1.3 Objectives and research question.....	2
1.4 Methodology	2
1.5 Outline	3
Chapter 2 - Literature review	5
2.1 Introduction	5
2.2 Blockchain technology.....	5
2.2.1 Scope.....	5
2.2.2 Blockchain structure	6
2.2.3 Blockchain types	8
2.2.4 Consensus mechanisms.....	9
2.2.5 Bitcoin blockchain	10
2.2.6 Ethereum blockchain.....	12
2.2.7 Security issues and challenges	18
2.3 Construction industry issues	20
2.3.1 Scope.....	20
2.3.2 Construction problems	21
2.4 Blockchain solutions for the construction industry.....	23
2.5 Summary	26

Chapter 3 - Information sharing platform	29
3.1 Introduction	29
3.2 Objective.....	29
3.3 Conceptual modal.....	30
3.3.1 Overview.....	30
3.3.2 Components	30
3.3.3 Features overview	33
3.4 Technical implementation.....	34
3.4.1 Overview.....	34
3.4.2 Architecture.....	34
3.4.3 Web server	36
3.4.4 Frontend.....	37
3.4.5 Backend server.....	38
3.4.6 Database.....	39
3.4.7 Blockchain	41
3.4.8 Smart contract	42
3.5 User interactions	45
3.5.1 Institution and admin user creation	47
3.5.2 Login.....	48
3.5.3 User management.....	49
3.5.4 Project management.....	50
3.5.5 File upload	52
3.5.6 File share	54
3.5.7 File signature request	55
3.5.8 File signing.....	57
3.5.9 File download.....	58
3.5.10 File deleting	59
3.5.11 File authenticity verification	59
3.6 Summary	62
Chapter 4 - Results and discussion.....	63
4.1 Introduction	63
4.2 Conceptual model	63
4.3 Proof-of-concept.....	64
4.4 Summary	68

Chapter 5 - Conclusions and future developments	69
5.1 Overview	69
5.2 General conclusions	69
5.3 Future developments	71
References	73

List of Tables

Table 1 – Comparison between blockchain types (adapted from Lastovetska (2019))	8
Table 2 – Block structure (adapted from Bitcoin (2019a)).....	10
Table 3 – Block header (adapted from Bitcoin (2019b)).....	11
Table 4 – Bitcoin denominations (Bitsusd, 2019)	12
Table 5 – Ether denominations (adapted from Ethereum (2019a))	18
Table 6 – Descriptive statistics of the transaction duration for different gas prices..	65
Table 7 – Transaction cost analysis (rate obtained for 26/09/2020 from Coinbase (2020)).....	67

List of Figures

Figure 1 – Relationship between blocks in blockchain (Nakamoto, 2008)	5
Figure 2 – Ethereum Ethash hashing function.....	7
Figure 3 – Connection between an application and the Ethereum network (Web3j, 2019)	13
Figure 4 – Ethereum raw transaction object	16
Figure 5 – Hash of the smart contract function and its inputs	16
Figure 6 – Soft fork (Investopedia, 2019b).....	19
Figure 7 – Hard fork (Investopedia, 2019a)	19
Figure 8 – Schematic diagram of collaborative information sharing applied to Integrated Project Delivery	23
Figure 9 – Relation between central authority and institutions	31
Figure 10 – Relation between users, projects and files of an institution	32
Figure 11 – File interactions	33
Figure 12 – Architecture diagram.....	35
Figure 13 – Application architecture	36
Figure 14 – Backend server layers.....	38
Figure 15 – Database entity-relationship diagram.....	40
Figure 16 – Smart contract function to register a file	43
Figure 17 – Smart contract function to retrieve file information.....	44
Figure 18 – Smart contract transactions	44
Figure 19 – Smart contract transaction details.....	45
Figure 20 – Architecture components interplay for each user interaction.....	46
Figure 21 – Login page	48
Figure 22 – Example of a JWT token generated after a successful login.....	48
Figure 23 – Application dashboard.....	49

Figure 24 – Empty dashboard	49
Figure 25 – User management dialog	50
Figure 26 – Project creation dialog	51
Figure 27 – Alert showing that the project was created with success	51
Figure 28 – Project edit or delete	52
Figure 29 – File upload dialog	53
Figure 30 – File upload processing	53
Figure 31 – Dashboard with an uploaded file	53
Figure 32 – Share file dialog	55
Figure 33 – Request signatures dialog	55
Figure 34 – Request signature processing	56
Figure 35 – File info with a pending signature	56
Figure 36 – Pending signature with the label	57
Figure 37 – Registration of a file signature processing	57
Figure 38 – Given signature with the label	58
Figure 39 – Given signature with the name of the user	58
Figure 40 – Download of a specific file version	59
Figure 41 – Confirmation dialog to delete a specific file version	59
Figure 42 – Verification page	60
Figure 43 – File uploading for verification	60
Figure 44 – File verification details	61
Figure 45 – Verification of an unregistered file	61
Figure 46 – Boxplot of the transaction duration for different gas prices	66
Figure 47 – Regression function of the transaction duration for different gas prices	67

Abbreviations

- API – Application Programming Interface
- BIM – Building Information Modelling
- CSS – Cascading Style Sheets
- DAO – Decentralized Autonomous Organization
- DOM – Document Object Model
- DTO – Data Transfer Object
- EVM – Ethereum Virtual Machine
- GDPR – General Data Protection Regulation
- HTML – Hypertext Markup Language
- HTTP – Hypertext Transfer Protocol
- IDE – Integrated Development Environment
- JDBC – Java Database Connectivity
- JPA – Java Persistence API
- JSON – JavaScript Object Notation
- JWT – JSON Web Token
- NPM – Node Package Manager
- POC – Proof-of-Concept
- REST – Representational State Transfer
- RPC – Remote Procedure Call
- SHA – Secure Hash Algorithm
- SPA – Single-Page Application
- UI – User Interface

Chapter 1 - Introduction

1.1 Scope

The technology industry is constantly evolving and the emergence of new technologies allows to solve problems that could not be easily solved.

Nowadays, one of the main technology trends is blockchain. This concept was introduced by the Bitcoin whitepaper (Nakamoto, 2008) and is characterized by being a database that stores transactions registries in an immutable, transparent and decentralized way. It is shared among network nodes, updated by the miners (network nodes that perform computational effort) and supervised by all users. Due to its characteristics, this technology is useful in contexts where there is no trust between parties or a need for intermediaries exists (Swan, 2015).

Studies about blockchain technology applied to the industry can be found in the literature: from chemistry (Sikorski *et al.*, 2017) to finance (Guo & Liang, 2016) or health (Mettler, 2016). However, only recent studies have integrated this technology with the construction industry (Das *et al.*, 2020; Yang *et al.*, 2020). In this way, there exists a wide opportunity to study this topic at a practical level.

1.2 Motivation

One of the challenges that the construction industry faces is the lack of trust among collaborators and information sharing processes (Wang *et al.*, 2017). This problem is related to the existence of a high information flow in medium to large construction projects that can lead to communication failures and, in extreme cases, to litigations or disputes.

Besides, the commercial solutions available in the market do not provide technological solutions to verify files' integrity (including non-conventional files, such as BIM or technical drawings) and the identity of who modified them, that can be used in case the software license is expired or the application ceases to exist.

This dissertation proposes a conceptual model and presents a proof-of-concept (POC) as a possible solution for the information sharing problem in the construction industry using blockchain technology.

1.3 Objectives and research question

The two main objectives of the current dissertation are:

1. Define a conceptual model for the application of blockchain technology to support information sharing in the construction industry;
2. Implement, demonstrate and evaluate a POC for the proposed model.

The research question is: how can blockchain technology support the construction industry within the scope of information sharing?

To answer this question, the following steps were established:

1. Analyse the construction industry problems that can be mitigated through technological solutions;
2. Analyse the blockchain technology usage as a mitigation for the identified problems;
3. Development of a conceptual model for a platform to support information sharing;
4. Development of a POC for the conceptual model;
5. Evaluation of the model as a solution for the identified problem.

1.4 Methodology

A methodology approach is defined to answer the research question. The first two steps will be addressed through the literature review specialized in i) construction challenges that can be mitigated through technological solutions and ii) in blockchain technology, including the several ecosystems on top of it (*e.g.* Bitcoin and Ethereum). For that, scientific papers and whitepaper references will be employed.

In the third and fourth steps, one of the previously identified construction industry problems will be chosen and a conceptual model and its practical implementation as a POC will be detailed. All relevant concepts will be defined and the implementation will be explained in detail.

The last step will be accomplished by evaluating how the proposed model solves the identified challenge.

1.5 Outline

The current dissertation is composed of five chapters briefly described as follows:

- Chapter 1, the current chapter, introduces the topics of the dissertation, explains the motivation and presents the research question. It also describes the methodology and the current outline.
- Chapter 2 presents the literature review about blockchain technology and construction industry problems.
- Chapter 3 describes the proposed solution to minimize one of the construction industry issues identified in the previous chapter. It presents the conceptual model and details the technical implementation of the POC, detailing the libraries, frameworks and programming tools chosen in the implementation phase. User interactions with the application are also illustrated.
- Chapter 4 presents a discussion about the conceptual model and the evaluation results for the POC.
- Chapter 5 shows an overview of the current work and general conclusions. This chapter also discusses paths for future developments.

Chapter 2 - Literature review

2.1 Introduction

The current chapter presents the literature review about blockchain technology and construction industry problems.

The blockchain technology is presented, including the two most known public networks: Bitcoin and Ethereum. Major security issues and challenges are also introduced.

The construction issues are discussed and their possible solutions with blockchain technology are presented.

2.2 Blockchain technology

2.2.1 Scope

The blockchain concept was proposed by Satoshi Nakamoto with the Bitcoin whitepaper (Nakamoto, 2008). This technology allowed the first creation of a virtual currency that is not regulated by a central authority (Tapscott & Tapscott, 2016).

Blockchain is a digital ledger that records all transactions that occur within a network inside *blocks*. In general, each *block* contains a reference to the previous block, a set of transactions and a *nonce* (these concepts will be discussed in sections 2.2.5 and 2.2.6) (see Figure 1).

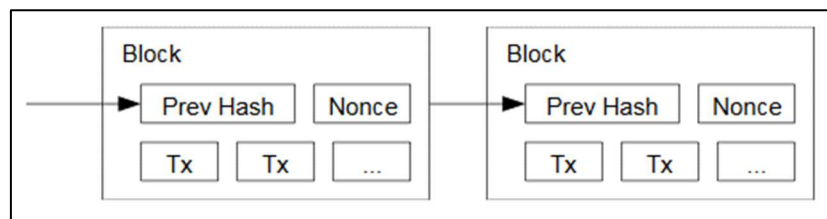


Figure 1 – Relationship between blocks in blockchain (Nakamoto, 2008)

The ledger is maintained by a set of nodes called *miners* composing a *peer-to-peer* network. The role of the *miners* is to verify the authenticity of each transaction by solving cryptographic algorithms.

Each node has a full copy of all records in the ledger since the first block, called the *genesis* block. After solving the cryptographic algorithms, the authentic transactions are broadcast to other *miners* and, if verified, broadcast to the network to keep all nodes up to date.

The regulation of each transaction is done by a consensus mechanism between the *miners*, removing the need for trust between the users (Swan, 2015).

The usage of blockchain as a core technology allowed the solution of technical challenges that prevented the creation of virtual coins. The two major problems were the Double-Spending and the Byzantine Generals' Problem (Swan, 2015). The first is related to the ability of a virtual coin to be copied, allowing the user to spend more than he holds (Hoepman, 2010), while the second is related to the lack of trust in the transactions between two parties (Lamport *et al.*, 1982).

Besides blockchain being born in a virtual coin context, an extrapolation to other applications quickly took place, resulting in various scopes: *Blockchain 1.0*, *2.0* and *3.0*. *Blockchain 1.0* is related to virtual coins and their economic power. The second is associated with contracts and assets that can be transacted. The latter is related to future applications in governance, health, science and culture (Swan, 2015).

The information recorded on the blockchain is by nature immutable and a decentralized source of truth and trust. These characteristics are the ones that allow this technology to be applied in very different industries, particularly in contexts where an agreement is needed between two parties that do not trust each other.

2.2.2 Blockchain structure

In general, blockchains are composed of a common structure. However, some blockchains have specific elements in their implementation that allow specific features.

The basic common element is the *address*, which is typically derived from the public key and is used as a unique identifier of each specific client (Bashir, 2017). *Wallets* are used to store addresses and their corresponding private keys.

The *transaction* is the main component of the blockchain, being responsible for the information transfer between addresses.

The *block* is the basic unit of the ledger. Each block typically contains multiple transactions and some elements as the previous block cryptographic hash, the timestamp (a representation of the moment the block is generated) and the *nonce* (the value used to adjust the cryptographic hash of the block according to the consensus algorithm) (Bashir, 2017). In Bitcoin, the block size is currently 1 MB (Bitcoin, 2019c).

A *node* is a device that is connected to the blockchain network. In general, nodes can have several functions depending on the blockchain type, such as keeping a copy of the blockchain and processing transactions (Bashir, 2017).

A *hash* is a cryptographic function that transforms input data into a fixed-length string (see Figure 2) and is intended to uniquely identify the corresponding input data. Hash collisions should be non-existent or extremely rare. This function is responsible for keeping the reliability and integrity of the ledger by not accepting fraudulent data or transactions. The hashes are used to represent the state of the blockchain since every new input is linked to the previous one and, therefore, influenced by all previous transactions that have occurred on the blockchain.

A hashing function has some properties (Lisk, 2019):

- It is not possible to produce the same hash for different inputs;
- Always generates the same output for the same input;
- The hash generation should be quick;
- It is extremely difficult to determine the input based on the output;
- A small change in the input produces a completely different output.

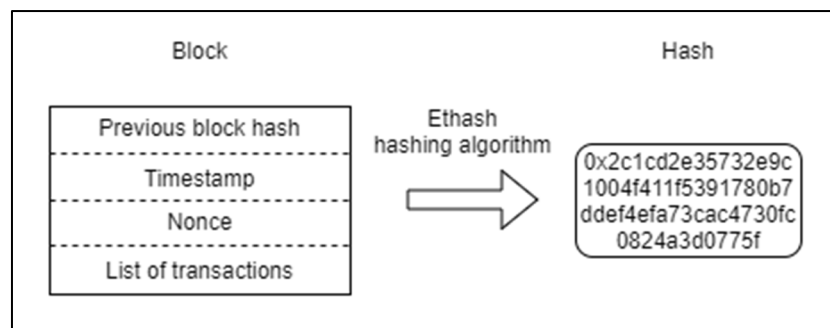


Figure 2 – Ethereum Ethash hashing function

Typically, the working process of a blockchain interaction starts with a client creating a *transaction* to send some data to an *address*. The *transaction* is signed with the private key of the client account and sent to a local *node*, which broadcasts it to its peers that validate the transaction based on pre-defined conditions. Once the *transaction* is validated, it is included in a *block* and the *miners* can start applying a consensus mechanism. If the *block* is considered valid and a consensus is reached, then the block is propagated to the network and all *nodes* will update the blockchain with the newly created *block* (Bashir, 2017).

Certain blockchains, like Ethereum, have a virtual machine that allows Turing Complete code to run on it. This allows the development of autonomous programs that will execute on the blockchain, called smart contracts, which are executed under specific conditions. Smart contracts allow the business logic encapsulation and the decentralization of code execution (Bashir, 2017).

Hyperledger is a project initiated by the Linux foundation focused on blockchain technology. The project aims to build an open source distributed ledger framework that can be used to build applications that support global business transactions (Bashir, 2017).

2.2.3 Blockchain types

Blockchain technology can, in general, be divided into three major categories: public, consortium and private blockchains (Lin & Liao, 2017). Table 1 shows a comparison between these three types.

Table 1 – Comparison between blockchain types (adapted from Lastovetska (2019))

Property	Blockchain types		
	Public	Consortium	Private
Consensus determination	All miners	Selected set of nodes	Within one organization
Read permission	Public	Public or restricted	Public or restricted
Immutability level	Almost impossible to tamper	Could be tampered	Could be tampered
Resources usage	High	Low	Low
Centralization	No	Partial	Yes
Consensus process	Permissionless	Needs permission	Needs permission

Public blockchains are publicly open and everyone can see the transactions and participate as a node in the consensus process. Node participants can be rewarded for their effort during the consensus mechanism. This type of ledger is not owned by a specific person, instead is maintained by all participants that maintain a copy of the ledger on their nodes. Examples of this type of blockchain are Bitcoin and Ethereum (Bashir, 2017).

A consortium blockchain is a type where the subset of the nodes that have the authority to validate the transactions is previously decided. This type is usually used by partnerships and its data can either be public or private. An example of this type is a blockchain created with Hyperledger software (Lin & Liao, 2017).

A private blockchain is characterised by not every node being allowed to participate in the blockchain and having strict data access (Lin & Liao, 2017). This type of blockchain is mainly used for database management and auditing.

2.2.4 Consensus mechanisms

The consensus is a process that allows an agreement between untrusted *nodes* in a data state. To achieve consensus between multiple nodes in a distributed system a consensus algorithm must be used.

A consensus mechanism is a set of actions that are carried out by nodes to reach a state agreement. Some requirements are needed to have the desired result of a consensus mechanism: all honest nodes agree on the same value and can terminate the consensus algorithm; the agreed value must be the same as the value proposed by at least one honest node; the consensus algorithm should be resilient in the presence of faulty or malicious nodes; each node only vote once in each consensus cycle (Bashir, 2017).

The consensus mechanism verifies that the latest block was correctly added to the blockchain and that the message that was stored by the node is correct. This protects the blockchain from malicious attacks, such as a hard fork attack (Lin & Liao, 2017).

The two most common consensus mechanisms are Proof of Work and Proof of Stake. However, some variants exist such as Delegated Proof of Stake.

Proof of Work is a type of consensus mechanism that uses computational resources to prove that enough effort has been done before proposing a value for acceptance by the network. This type of consensus is characterised by its difficulty to generate data, involving several trial and computations, but is easy to be verified by others. One of the drawbacks of this mechanism is the demand for a lot of computational power and, therefore, electricity. This type is currently being used by Bitcoin and Ethereum (Bashir, 2017).

Proof of Stake is a consensus mechanism that selects the next block creator through a combination of random selection and the amount of cryptocurrency that a user holds or

the time that has been holding it. This consensus type has as main advantages the considerable reduction of energy needed and the increased security when compared with Proof of Work. The increase of security comes from the attackers having to hold a considerable stake in the network, which will lead to bigger consequences to their assets (Lin & Liao, 2017). Ethereum is planning to adopt this type of consensus (Buterin, 2019).

The two most known public blockchains are Bitcoin and Ethereum, which will be detailed in the following sections.

2.2.5 Bitcoin blockchain

The generation of new blocks in the Bitcoin blockchain occurs approximately every ten minutes. This is ensured by the network difficulty that is calculated every 2016 blocks based on the time that took to generate those previous blocks. At the desired rate, the generation of 2016 blocks takes two weeks. If those previous blocks validation took more than 2 weeks the difficulty is decreased, while if it took less the difficulty is increased. The difficulty is converted into the network hash rate, which is the *mining* “speed” of the network (Bitcoin, 2019e; Murabito, 2019).

In Bitcoin, each block contains, among other things, the current timestamp, a set of transactions, a reference to the previous block and a nonce. The nonce is a 4-byte field and its value is adjusted by the miners to regulate the hash of the block with the current target of the network, which is defined by the network difficulty (Bitcoin, 2019d). Table 2 presents the Bitcoin block structure and Table 3 shows the details of each block header.

Table 2 – Block structure (adapted from Bitcoin (2019a))

Field	Description	Size
Magic no.	Value always 0xD9B4BEF9	4 bytes
Blocksize	Number of bytes until the end of the block	4 bytes
Blockheader	Consists of 6 items	80 bytes
Transaction counter	Positive integer (Varint)	1 – 9 bytes
Transactions	List of transactions	Transaction counter

Table 3 – Block header (adapted from Bitcoin (2019b))

Field	Purpose	When updated	Size (bytes)
Version	Block version number	Software is updated	4
hashPrevBlock	256-bit hash of the previous block header	There is a new block	32
hashMerkleRoot	256-bit hash based on all transactions in the block	Transaction is accepted	32
Time	Current block timestamp in seconds since <i>epoch</i>	Every few seconds	4
Bits	Current network target	Network difficulty is adjusted	4
Nonce	32-bit number (starts at 0)	A hash is evaluated	4

For a given network mining difficulty, a set of leading zeros in a block hash is needed to consider the block as valid. The exact number of zeros depends on the network hashing power at each moment. If the hashing power increases, the number of zeros increases, and vice-versa (Murabito, 2019).

In order to have a block hash with the needed set of zeros, the parameter *nonce* is changed (all the other block components are constant except the current timestamp). When a *miner* gets a block hash with the desired number of zeros, the block is signed and broadcast to other *miners* to verify its authenticity and, if consensus is reached, the block is added to the blockchain and broadcasted to all nodes. This is the reason why an average block signature takes ten minutes: to assure that the information is broadcasted and all nodes have the same information. The miner that signs the block gets a reward for its computational effort (Lin & Liao, 2017; Murabito, 2019).

Bashir (2017) states that, in Bitcoin, a good practice is the generation of a new address by the user for each transaction to avoid transactions being linked to a common user.

Table 4 shows the Bitcoin denominations, varying from Satoshi, the smallest, up to the Megabit, the largest unit.

Table 4 – Bitcoin denominations (Bitsusd, 2019)

Denomination	Abbreviation	Familiar name	Value in BTC
Satoshi	SAT	Satoshi	1E-8
Microbit	μBTC	Microbitcoin or Bit	1E-6
Millibit	mBTC	Millibitcoin	0.001
Centibit	cBTC	Centibitcoin	0.01
Decibit	dBTC	Decibitcoin	0.1
Bitcoin	BTC	Bitcoin	1
DecaBit	daBTC	Decabitcoin	10
Hectobit	hBTC	Hectobitcoin	100
Kilobit	kBTC	Kilobitcoin	1000
Megabit	MBTC	Megabitcoin	1E6

2.2.6 Ethereum blockchain

The Ethereum platform is based on an alternative protocol to develop decentralized applications. It offers a layer of abstraction that contains at its basis a blockchain that has a Turing Complete programming language (Buterin, 2014).

The platform allows the development of decentralized applications and smart contracts with desired rules. Smart contracts are programs running into the blockchain, storing information, processing inputs and writing outputs (Bogner *et al.*, 2016).

The Ethereum network can be divided into three types: mainnet, testnet and private net. The mainnet is the current network of Ethereum, while the testnet is used to test smart contracts and applications before deploying to the mainnet. Ropsten and Rinkeby are examples of a testnet. The private net is a private network that can be used as a permissioned blockchain.

Ethereum has an associated virtual coin, *Ether* (ETH), and permits decentralized applications to be processed by several network nodes. This allows the applications to be always available (Wood, 2017).

Ethereum network

The Ethereum blockchain connection is composed of a peer-to-peer Ethereum network, Ethereum clients running on the network nodes and a communication protocol (see Figure 3). An Ethereum client is responsible for the connection between an application and the Ethereum network.

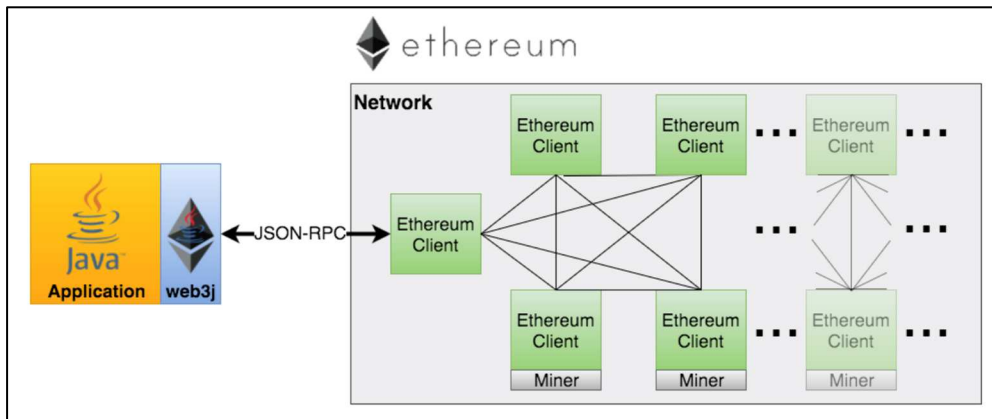


Figure 3 – Connection between an application and the Ethereum network (Web3j, 2019)

There are several implementations for Ethereum clients. However, the most common are go-Ethereum and Parity. Go-Ethereum, also known as Geth, is the Go implementation while Parity is the Rust implementation. Some light clients, called Simple Payment Verification, only download a subset of the blockchain. These clients are used in devices with low resources and are used to verify transactions (Bashir, 2017).

Ethereum account

The Ethereum state is composed of objects designated by accounts, having each a 20-byte Ethereum address. The transfer of value or information between them is achieved through state transitions. In general, there are two types of accounts: externally owned account and contract account. The first allows sending messages to other accounts by creating and signing a transaction. The second has a contract code, called a smart contract, that is executed when a message is received. An Ethereum account has four fields (Ethereum, 2020b):

- Nonce – a counter to avoid the same transaction to be processed more than once;
- Ether balance – the balance of the account;
- Contract code – only available if the account is a contract account;
- Storage – the storage of the account, empty by default.

An Ethereum wallet can be used to manage multiple Ethereum accounts. They can be of the following types: Smart Contract, Hardware, Mobile, Desktop or Web (EthHub, 2020).

Smart contract

The Ethereum blockchain allows the development of smart contracts, which are digital contracts that are developed in a high-level programming language, compiled into bytecode and deployed on the network to be executed by the Ethereum Virtual Machine (EVM) (Antonopoulos & Wood, 2018; Ethereum, 2019c).

The smart contract concept was created by Nick Szabo (Szabo, 1997) and is a user-defined computer program with specific rules that are automatically executed and enforced without mediation. It manages its internal state using a state machine model (Bashir, 2017).

An Ethereum smart contract is an autonomous program that executes a specific function when queried by a message or transaction, controlling the key/value store to keep track of persistent variables. The execution of code can be to read and write to internal storage, send messages to other accounts or create other contracts (Ethereum, 2020b).

A smart contract can be written in four languages (Bashir, 2017):

- Mutan – Go-style language (not currently used);
- LLL – Lisp-like language (not currently used);
- Serpent - Python-like language;
- Solidity – Object-oriented and high-level programming language (influenced by C++, Python and JavaScript).

There are also other experimental languages being developed and improved like Vyper, which is a strongly-typed Python language, and JULIA, a language for Ethereum virtual machines (Bashir, 2017; Ethereum, 2019c).

Ethereum transaction

Each transaction execution is subjected to a fee, called *gas*, that is tabulated according to the involved computational effort. It aims to support the cost of the computation performed. Each transaction specifies the amount of *gas* that is willing to spend for the operation, the *gas* limit. If the *gas* limit is reached before the end of the operation, then a *run out of gas* error is thrown and the transaction is rolled back. If the operation

successfully finishes the remaining *gas* is refunded to the transaction sender (Bashir, 2017).

A transaction in Ethereum is data signed digitally using a private key and can be of two types: message calls and contract creation. The first is used to pass data from one account to another, while the second results in new contract creation. Both transactions are composed of common fields (Bashir, 2017):

- *nonce* – a number that is incremented by one for each transaction sent. It acts as a unique identifier for the transaction, preventing double-spending. This field forces the transactions to be in order and prevents skipping. This means that a transaction with nonce 1 cannot be mined before another with nonce 0 (Bashir, 2017; MyEtherWallet, 2019);
- *gasPrice* – represents the price per unit of *gas* (in Gwei) that the transaction sender is willing to pay for the transaction;
- *gasLimit* – or simply *gas*, indicates the maximum amount of *gas* that can be used during the transaction execution. This value represents the maximum amount that a user is willing to pay for the transaction computation. It prevents the transaction sender account from being drained;
- *to* – the address of the transaction recipient;
- *value* – the number of Wei (smallest Ether unit) to be sent to the recipient;
- *signature* – the signature of the transaction sender;
- *init or data* – the *init* field is used in contract creation transactions while *data* is used in message call transactions. The first represents a byte array that specifies the EVM code that will be used during the account initialization (getting destroyed after), while the second is related to the input data of the message call (Bashir, 2017).

Smart contract interaction

The lifecycle of an Ethereum transaction for a smart contract message call is explained in the current section according to Ethereum (2020a).

A transaction needs to be created to invoke a function in a smart contract to change its state. To start an Ethereum transaction the user should be connected to an Ethereum client, Geth or Parity node, which is connected to a network (either mainnet or testnet).

The Ethereum network has both miner and non-miner nodes. The miner nodes maintain a transaction pool where the transactions are sorted by the *gas* price before being mined. Since the common node configuration is to look for transactions with higher *gas* prices, these transactions are more likely to be mined first (*i.e.* to be included in the next block).

The amount of transactions within the transaction pool is finite. This means that if a transaction has a lower *gas* price there is a probability that it will never be mined. In these cases, the transaction might need to be rebroadcast with a higher *gas* price. To do this the *nonce* field should be the same to override the previous transaction.

The sum of the *gas* limit of all transactions must not exceed the block *gas* limit. Eventually, a miner selects the transaction to include in the next block with other transactions, which after being validated are included in a pending block and the Proof of Work consensus algorithm starts.

When a miner node finds a valid block, it is added to the blockchain and is broadcast to other nodes. All nodes will receive, verify and sync the new block with the local copy of the blockchain by executing all the transactions in the block.

Decentralized Autonomous Organizations

Ethereum allows the setup of a decentralized organization, which is a set of smart contracts that run on the blockchain. When the business logic is executed by artificial intelligence, rather than by human input, it is referred to as a Decentralized Autonomous Organization (DAO). The governing entity of a DAO is enforced by code instead of humans. However, there exists a person that acts as a maintainer and a proposal evaluator for the community, called Curator. A DAO can even contract external contractors when required by the token holders (Bashir, 2017).

One of the most famous DAO projects is *The DAO* which aims to be a venture capital fund platform, acting as a decentralized business model without an owner. However, it was hacked by exploring vulnerabilities in its code, draining a considerable amount of Ether into a child DAO. The community decided to perform a hard fork of the Ethereum blockchain to recover the stolen funds. However, some elements complained because this decision goes against the nature of the Ethereum itself, and preferred to keep mining the previous version, now known as Ethereum Classic (Bashir, 2017).

Ether denominations

Ethereum has the following metric system of denominations, which are used to measure units of Ether. The smallest is called Wei (see Table 5).

Table 5 – Ether denominations (adapted from Ethereum (2019a))

Unit	Denomination	Wei value
wei	Wei	1
Kwei	Babbage	1000
Mwei	Lovelace	1E6
Gwei	Shannon	1E9
microether	Szabo	1E12
milliether	Finney	1E15
ether	Ether	1E18

2.2.7 Security issues and challenges

The increase in blockchain adoption revealed some problems and challenges. According to Lin & Liao (2017), some of the identified problems are:

- Majority attack;
- Fork problems;
- Scalability;
- Confirmation time of the blockchain data;
- Regulatory problems;
- Integrated cost problem.

The majority attack is related to having the majority of the nodes being controlled by the same entity. By controlling more than 51% of the computing power the entity can control which blocks are permissible and modify the transaction data.

Fork problems are related to different versions of the nodes and agreements regarding versions upgrade, which may include different consensus rules. This leads to the existence, in the same network, of new and old nodes.

Forks can be of two types: soft and hard. In a soft fork, a software change invalidates blocks and transactions that previously were considered valid. This type of fork is backward-compatible as the new nodes recognize the new blocks and transactions as valid, only requiring the majority of the miners to upgrade to the new software (Figure 6).

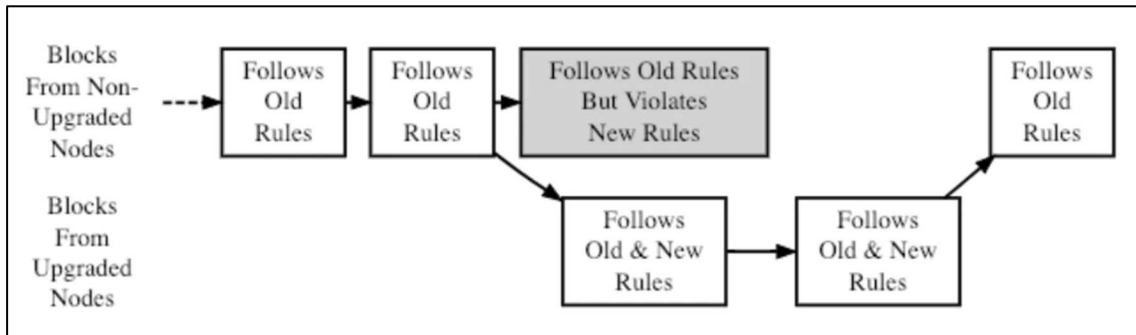


Figure 6 – Soft fork (Investopedia, 2019b)

The hard fork is related to a big change in the software that leads to previously invalid blocks being considered valid (or vice-versa). As this type is not backward compatible, all nodes must upgrade to the new version of the software. A hard fork creates a fork in the blockchain, having a new ledger with the upgraded version and an old one with the previous version (Figure 7).

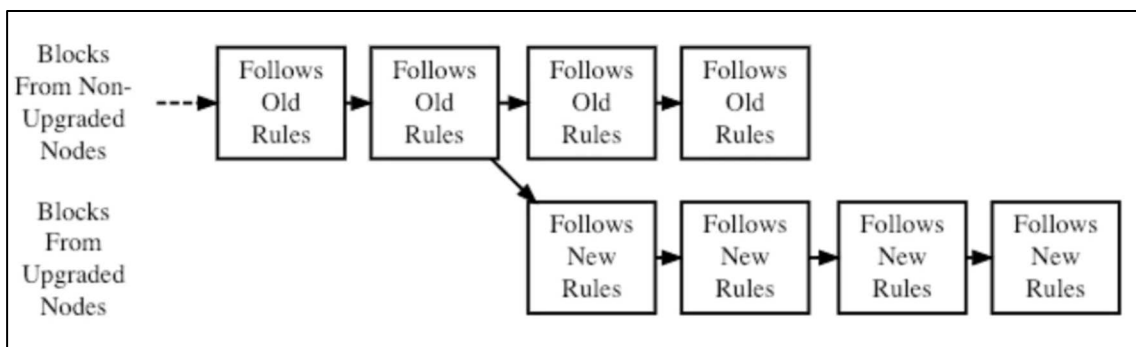


Figure 7 – Hard fork (Investopedia, 2019a)

Scalability is a problem because blockchain data becomes bigger when a new block is added, leading to an increase in the need for more storage and computing power. With the increase of the blockchain, more time is needed to synchronize data.

Besides the time needed to confirm blockchain data being less than the normal banking system transactions, it sometimes can take more time than what is desired to explore new blockchain applications, such as with the Internet of Things applications.

The lack of regulation leads to a generalized lack of trust in blockchain technology. Incidents such as *The DAO* hack evidenced that the non-existence of regulation sometimes can have unwanted effects.

The adoption of blockchain technology forces companies and the public sector to invest in the replacement of old systems. This involves initial costs to support the infrastructure for the new technology.

2.3 Construction industry issues

2.3.1 Scope

The construction industry is one of the key industries in all countries, and one of the largest in the world economy. However, its productivity has nearly stagnated, with a labour-productivity average growth of 1% a year over the last two decades, while in manufacturing it was 3.6% and the total world economy has grown 2.8% yearly (Barbosa *et al.*, 2017). One of the reasons for the lack of productivity is the fact that this is one of the latest industries adopting technology and innovation (Belle, 2017; Barbosa *et al.*, 2017).

Construction is an industry that typically involves several participants, processes and products, where the projects are executed by many companies (Heiskanen, 2017; Wang *et al.*, 2017). In general, in large developments, dozens of companies are involved in the multiple components of the designs (structures, landscaping, architecture, electricity, mechanical equipment, water and residues, etc...) and execution. Commonly, each involved company contracts other contractors, leading to a cascade of contracts.

Studies (Black, Akintoye, & Fitzgerald, 2000, Wong & Cheung, 2005) have shown that end-customer satisfaction can be increased by partnering. One of the key elements for partnering success between all involved participants in construction is trust, which can increase with performance growth.

According to Penzes (2018), one of the most valuable and intangible assets in every business between organizations and partners or customers is trust. Trust is enabled by third parties and intermediaries ensuring that all parties have a legal right, authority and transparency to do business with each other. However, with the increase of business volume, the trust and transparency given by third parties become too distant and complex.

This complexity leads to missing information, time consumption and additional costs for all parties.

2.3.2 Construction problems

Information sharing

The high number of participants typically involved in a large construction project leads to high information flow between parties. This often results in communication failures, litigations and time waste, leading to projects rarely being developed as planned (Heiskanen, 2017).

Besides information sharing issues, Wang *et al.* (2017) refer that the construction industry has several challenges in terms of trust and process automation, and state that, with the increase of the construction project complexity, the hiring of international labour is growing. This leads to a need for a clear and honest way to share information, that is not facilitated by conventional contracts and information sharing mechanisms.

Building Information Modelling (BIM) is a digital representation of the physical and functional characteristics of a building and can be used as an information sharing system (Sacks *et al.*, 2018). However, Turk & Klinc (2017) point out that its adoption is diffculted by several problems, among them the existence of several BIM models for the same building, the difficulty of sharing the models and the lack of a mechanism that tracks all model changes in a reliable way, which can be used as responsibility mechanism.

Currently, exists in the market solutions to address information sharing problems in the construction industry, examples of these applications are Asite (2020), Autodesk (2020) and Trimble (2020). These applications incorporate a wide variety of features, such as file pre-visualization, the possibility to review and add comments on shared files or allowing to interact with drawings. However, to the author's knowledge, they all have expensive licenses and lack file integrity verification mechanisms that can be used by everyone, even in case the license has expired or the platforms no longer exist.

Delayed payments

Another problem is the delayed payments or non-payments, which are frequently identified as one of the main sources of business failures and related to an increase in disputes (Wang *et al.*, 2017). According to the Euler Hermes report (Hermes, 2016) construction industry was the sector with more payment delays in the United Kingdom in

2015, being accounted for 31% of all reported incidents. The same report states that overdue payments are common in the United Kingdom and that the sector suffered a 27% year-on-year increase of overdue payments incidents in 2015. This leads to long disputes and delivery delays.

For United Kingdom construction companies, the average payment time in 2015 was 82 days and sometimes it can reach up to 120 days. This can cause a huge risk to all supply chains, especially for smaller companies that cannot support continuous payment delays without a robust financial situation (Penzes, 2018).

Among the current mechanism to prevent delayed payments are contracts, banking guarantees and insurances. However, these are inefficient as the current literature review reveals.

Construction contracts

One of the reasons for the high degree of information loss and level of conflict is the contract model. There are three dominant types of contracts in construction: Design-Bid-Build, Design-Build and Construction Management at Risk. There is a fourth method that is becoming more popular with the use of BIM: Integrated Project Delivery (Sacks *et al.*, 2018).

In the classic Design-Bid-Build, two different bids are placed. The first involves the design, the second is based on the result of the first one and involves the construction. Both the design and construction bidding involves many individuals and companies, with obvious technical, trust and contractual difficulties.

In the Design-Build method, the responsibility for the design and construction is consolidated into a single bid and entity. This decreases the number of technical and contractual interactions. However, even in the same company or consortium, there are competing interests and opportunities for misunderstanding.

In the Construction Management at Risk model, a construction manager is hired by the owner to provide a complete solution, design and building, at a guaranteed maximum price. The construction manager, which acts on the behalf and interest of the owner, may or may not be involved in the design and construction. Often these contracts include incentive clauses in which the construction manager and owner split the positive

difference between guaranteed maximum price and actual cost. If that difference is negative, the construction manager incurs financial penalties.

There are variations of these methods, including models where the building, or facility, is managed by the contractor for several years (usually more than 10). This has the advantage of eliminating or at least reducing difficulties in the handover.

Integrated Project Delivery is a more sophisticated model that aims to promote effective collaboration between owners, designers and contractors during all phases of the lifecycle. This model demands contracts that define mutual gains to all parties when the project aims are attained, thus aligning the interests in a common goal. The number of information exchanges increases exponentially in this model, as the flow of information is no longer linear, but circular among all parties. Collaborative tools that allow these information exchanges in a safe, reliable and traceable manner are needed (see Figure 8).

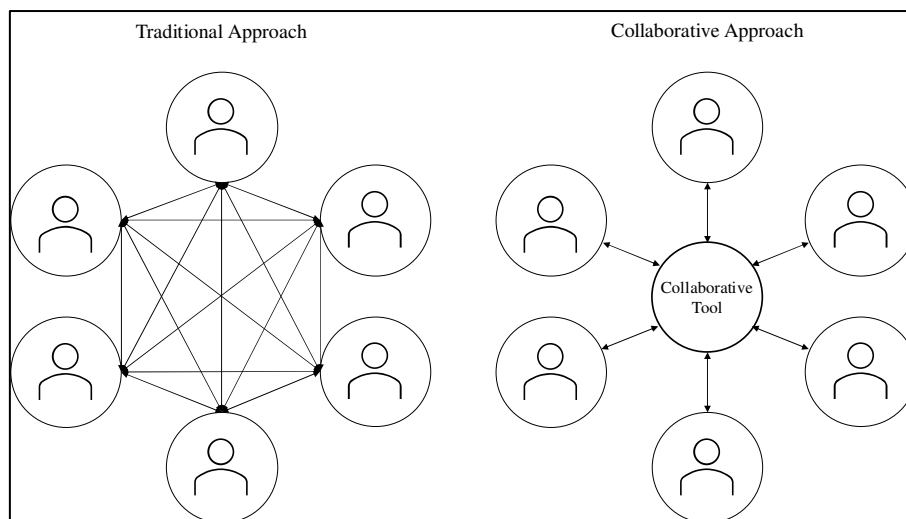


Figure 8 – Schematic diagram of collaborative information sharing applied to Integrated Project Delivery

2.4 Blockchain solutions for the construction industry

Blockchain offers a solution that allows a transparent distribution of information among all network participants in an immutable and simultaneous way where no party holds overall data control. This is important to an industry that needs a high level of trust in relationships.

Belle (2017) states that one of the main benefits of blockchain is the decrease in administrative costs due to digitalization and, in particular, the cost of licensing the

copyright. For Turk & Klinc (2017) the major benefits are having reliable construction logbooks, information about performed work and material quantities recorded during the construction phase, and sensitive information secure storage, such as sensor data, during the building occupation and maintenance phase.

The usage of blockchain technology as a basis for activities related to construction could remove the need for a personal trust relationship between the involved participants, by only trusting the information at the blockchain.

Even though the construction industry is an industry where the technology adoption is, typically, carried out slowly and gradually (Belle, 2017), some studies reveal that the new technologies and, in particular, blockchain can contribute to mitigate some of the identified problems, as detailed below. Examples of these studies are Mathews *et al.* (2017), Singh & Ashuri (2019) and Wang *et al.* (2017).

Information sharing

Blockchain can be used to reduce information sharing problems between the stakeholders by being the underlying technology for a platform that can collect inputs from all involved parties, recording every necessary interaction.

Penzes (2018) suggests that during the design phase the platform can record digital signatures of the design packages or calculations and their approvals and quality assurance steps without storing the original files. This can be an important point because of data protection policies. The same author states that these inputs can be used to track project progress and be used for accountability and traceability of design approvals because the underneath system is tamper-proof.

For the construction stage, the previous author refers that the platform can record information for each task regarding pre-defined specifications and quality assurance procedures that need to be followed, together with the team that is going to perform the assigned task. After the work is completed, the site engineer, together with the quality controller and project manager, verify and approve the performed work by digitally signing in the platform. This process automatically updates and displays construction progress to all parties.

The ability to share information between stakeholders within a collaborative platform can facilitate the management and monitorization of the construction project by reducing

contract misalignments and enabling collaboration between participants. With this platform, unforeseen issues that may result in claims and disputes can be solved faster and easier as secured records can be used for accountability. The platform also allows having transparent control over the quality of the provided services. This can allow companies to save time and money by avoiding construction delays due to unpredictable issues and achieving faster execution of the construction (Penzes, 2018; Singh & Ashuri, 2019).

The combination of BIM and Blockchain in one platform enables the BIM model to act as a single source of truth. This allows the addition of extra data to the model and ensures high accountability and traceability because of the irrefutable nature of the technology (Penzes, 2018).

The association between blockchain and BIM also allows keeping a trusted record about when and who made de changes to the model, serving as a basis for any legal argument that may occur (Turk & Klinc, 2017). This record allows the centralization of the information in one model and to share it between the participants, having a role and permission mechanism to read and write for each participant.

Acting as a single source of truth, a BIM model can aggregate information for the whole lifecycle, such as supply chain information, the provenance of materials and payment details (Penzes, 2018).

Mathews, Robles & Bowe (2017) studied the integration of BIM and blockchain to provide a collaborative platform as a possible solution for the trust problem. The authors suggested a private blockchain with network nodes for the involved stakeholders (*e.g.* client, architect, engineer and contractor) to record the collaborative interactions and rewarding individual contribution over his lifecycle through a cryptocurrency coin. It aims to bring together professionals creating a collaborative network and disrupting the existing top-down hierarchical structure that separates all involved collaborators.

Delayed payments

Wang *et al.* (2017) point as the main blockchain usages in this industry the ability to give final clients a transparent and traceable mechanism for the several materials used in the construction, elimination of time spent on document authenticity verification and the automation of payments. The authors state that this last point can be implemented through

smart contracts, where the final client pays to the main contractor and the payments to subcontractors are automatically triggered (*i.e.* a chained payment).

A blockchain-based platform can automate payments in the construction industry based on digitally approved work, contractual terms and smart contract actions. This platform can increase the transparency between all involved parties and ensure that the pre-defined terms agreed at the smart contract will always happen. The platform could be used to ensure that the payment will be done on time, as soon as all agreed criteria are met (Penzes, 2018).

Construction contracts

Construction contracts have evolved to promote greater collaboration and common objectives among those involved in all construction phases. They aim to decrease disputes and litigations, reducing time and costs. Inevitably, they require a higher flow of information and collaborative tools to handle it (Sacks *et al.*, 2018).

A blockchain solution involving information sharing and automatic payments can support modern contract types to deal with high information flows and automatically trigger payments as the construction is progressing. It can be used as an accountability mechanism in case of a dispute in any construction phase. This is especially important in larger constructions where the centralization of information plays an important role to avoid information loss and delayed payments.

2.5 Summary

In the present chapter, a literature review was performed about blockchain technology and the construction industry challenges.

Blockchain is a digital ledger that allows recording information in an immutable, transparent and decentralized way. All information is stored within blocks, with each block linked to the previous one, forming together a network.

There are three types of blockchain networks: public, consortium or private. Each of them can have their consensus algorithms, such as Proof of Work and Proof of Stake.

Were identified three types of construction problems that can be addressed with blockchain technology: lack of an accountable mechanism for information sharing, delayed payments and modern contract types.

For information sharing issues, blockchain can be used to record the interactions between involves parties, as design packages or BIM model files, that can later be used in case of disputes or litigations. The delayed payments can be mitigated by using blockchain to trigger automatic payments based on digitally approved work. This technology can also support modern contract types to deal with high information flows by supporting information sharing and automatic payments.

Chapter 3 - Information sharing platform

3.1 Introduction

The current dissertation proposes a platform to support information sharing in construction, overcoming one of the industry problems identified in section 2.3.2.

A conceptual model is proposed and a POC presented. The POC was developed to evaluate the integration between blockchain technology and the construction industry. It aims to answer the research question by evaluating if information sharing in the construction industry can be supported by blockchain technology.

The present chapter describes the proposed information sharing platform. Firstly, its objective is presented, followed by the presentation of the conceptual model including the description of the major features.

Secondly, the technical solution of the POC is presented, including an overview of the application architecture and its components, describing the selected libraries, frameworks and programming tools. The chosen web server, frontend, backend layers, database model and smart contract considerations are also detailed.

Lastly, the user interactions with the application are presented by illustrating each interaction with images of the application.

3.2 Objective

The platform aims to support construction management by facilitating information sharing between all involved participants in all construction phases. It intends to be the source of information for all the construction files, simulating a physical file sharing with signature records that can later be verified and used as an integrity verification mechanism.

The application should provide three major features to support the information sharing problem identified in section 2.3.2:

- Allow to share files, including technical drawings, contracts and BIM files;
- Provide a file integrity verification that can be used by anyone that has access to the file;
- Provide an identity mechanism that allows visualizing the file progression, with the identification of who modified or signed each version of the file.

3.3 Conceptual modal

3.3.1 Overview

The application is a central source of information, with files and all their versions, that allows users to upload files, or a new version of an existing file, download it, share it with other users, request signatures from other users or sign it. The involved participants can visualize the document progression and easily identify who modified or signed each version of the file.

The files uploaded to the application are stored in the database, while its cryptographic hash and the identity of who uploaded the file are recorded in a decentralized and immutable way in a smart contract on a public blockchain. The users that are requested to sign a version of a file and the ones that sign it are also recorded on the smart contract. This allows using the application to later verify if a file was registered on the platform and to identify who uploaded, requested to sign or signed it.

Since the files' information is stored in a public blockchain, they are immutable and tamperproof, which enables their integrity to be verified even in case the application ceases to exist. The information remains available outside of the application in every Ethereum node.

3.3.2 Components

The platform is supported by five main components: central authority, institutions, users, projects and files, defined as follows:

- Central authority is a single entity that is responsible for the application maintenance and to register institutions in the application;
- Institutions are representations of companies, entities or consortiums within the application. Each institution has an admin user role that can create users and projects in its institution and assign users to projects;
- Users are the collaborators of an institution with an account in the application;
- Projects are working groups that have collaborators from the same or different institutions working together;
- Files are documents that can be of three types: Technical Drawings, Contracts and BIM Files. Each file belongs to a specific project.

The central authority is responsible for the system maintenance and for companies, entities or consortiums identity verification, assuring that they are who they claim to be. After being successfully validated, the central authority creates an institution within the application with an admin user. The admin user is responsible for the institution management within the application, having the possibility to create users and projects (see Figure 9).

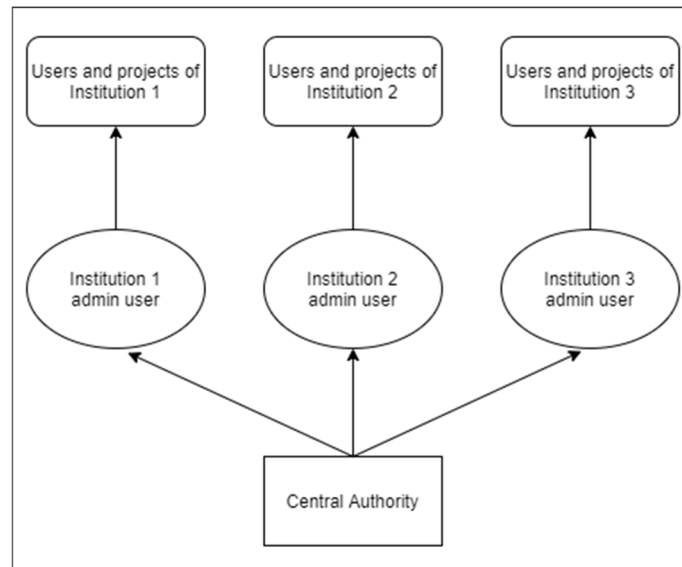


Figure 9 – Relation between central authority and institutions

An institution is registered on the application after being validated by the central authority. Its registration involves an admin user creation and a generation of an Ethereum wallet that is stored at the database, with the corresponding Ethereum address derived from its wallet. The institution and its admin user are also recorded in the smart contract, storing their name and Ethereum address. The latter is the link between the smart contract and the database. The creation of users within each institution is done by their admin user and follows an identical pattern, without the institution creation.

Admin users can create the projects of the institutions and assign their users to them. Each user can be assigned to more than one project. The projects and their access control are stored in the database. The platform access control is also performed at the database level. The users need an account associated with an institution to access the platform.

Figure 10 shows an example of an institution, designated by *Institution 1*, having one *admin user*, two *projects*, six *users* and twelve *files*. The admin user is responsible for projects' and users' creation within the institution and does not belong to any project. Users 1 and 3 are associated with both projects, while Users 2, 4, 5 and 6 are related to

only one. Each project has six files, divided into three categories: Technical Drawings, Contracts and BIM Files. Users only have access to the information explicitly shared with them within their projects, preventing them to have access to all files of the project. This information access control is performed at the database level, where each user has a list of files that can access.

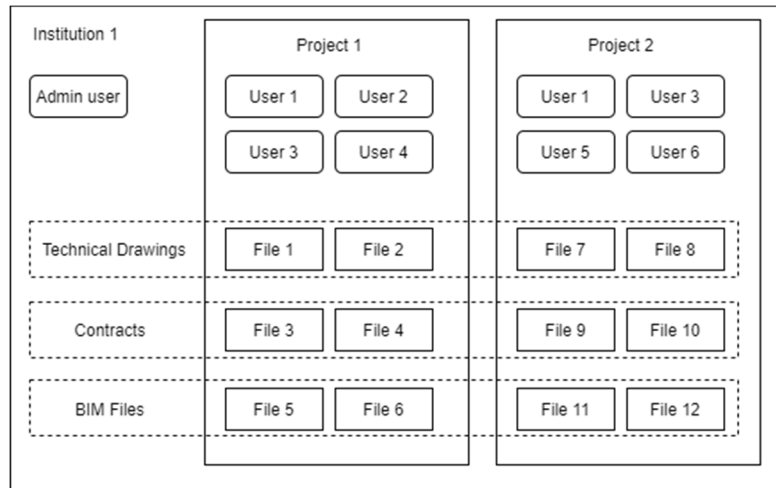


Figure 10 – Relation between users, projects and files of an institution

An institution can have several projects, one for each construction project. Within each project the information can be grouped by categories: Technical Drawings, Contracts or BIM Files. This mechanism allows users to easily organize the information within projects.

Files are organized by projects and categories. When a file is uploaded to the application it is only visible to the user that uploaded it, the file owner. To share it with other users of the project the owner can share directly or request a signature. The difference between both features is the registration on the blockchain, the first approach allows an additional user to access the file and does not involve blockchain while the second, besides allowing an additional user to access the file, registers on the smart contract that a signature request has been done. When a user signs a file, it is also registered on the smart contract that a signature was given.

Each file has a list of versions with several file modifications performed throughout the time, along with the date and identity of the user that modified it. Each file version has a list of signatures and signature requests. The signature requests are the pending signatures, while the signatures are the ones that were given to the file.

Figure 11 illustrates the user interactions with a file. In the example, users can interact with version 10 of File 1 that belongs to Project 1. This file version is owned by User 1, has two signature requests (Users 2 and 3) and one given signature (User 2). There is also User 4, with whom the file has been shared, that has access to the file without a signature request. The possible user interactions with this file version are the following: request signatures, sign, share, download or delete.

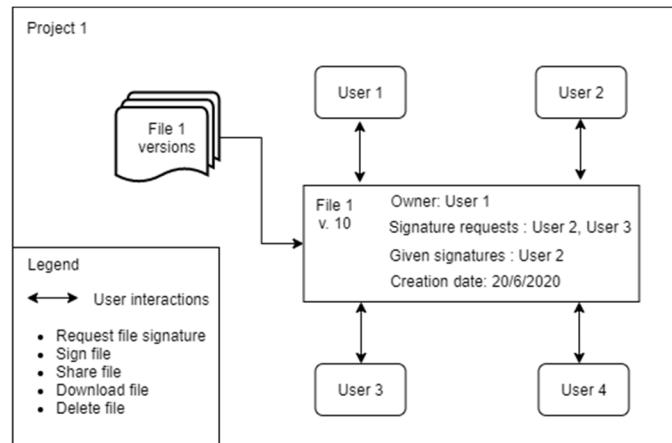


Figure 11 – File interactions

3.3.3 Features overview

The platform is divided into three types of users: central authority, admin user and normal users. Each type can perform the following interactions within the application:

1) Normal users

- Login into account;
- Visualise the files to which the user has access;
- Upload a new file (requires blockchain transaction);
- Upload a new version of an existing file (requires blockchain transaction);
- Share a file with a set of users within the project;
- Request a file signature from a set of users of the project (requires blockchain transaction);
- Sign a file that has requested his signature (requires blockchain transaction);
- List all versions of a file, the file history;
- Download a specific version of the file;
- Verify that a specific version of the file is authentic, its signatures and signatures requests (reads information from blockchain);

- 2) Admin users - all the previous plus:
 - Create users within its institution (requires blockchain transaction);
 - Create projects within its institution;
 - Associate its users to its projects.
- 3) Central authority
 - Create new institutions (requires blockchain transaction);
 - Create an admin user for each institution (requires blockchain transaction).

The platform is also accessible without an account to perform the file validation. This allows anyone to verify if a file was registered on the blockchain, its pending signature requests and who has signed it.

Some user interactions require a blockchain transaction to update the smart contract state. Since blockchain interactions are paid, the information stored on the blockchain should be the minimum as possible. However, the smart contract can be extended to record additional information.

3.4 Technical implementation

3.4.1 Overview

The platform is a web application, which nowadays is the standard in the development of software applications. It allows a continuous update of the application without any effort from the users. Each client (*e.g.* a user of the application) accesses the application through a web browser.

3.4.2 Architecture

A multi-tenant architecture was chosen because it allows having a central authority that verifies the identity of the institutional clients and to have a single point of contact with the blockchain. This implies that all clients of the platform connect to the same instance. The application has a software bundle composed of a web server, a backend server, a database and a blockchain node, that can either be on-premise, in the cloud or as a service (see Figure 12). The immutable information is stored in a smart contract on the blockchain.

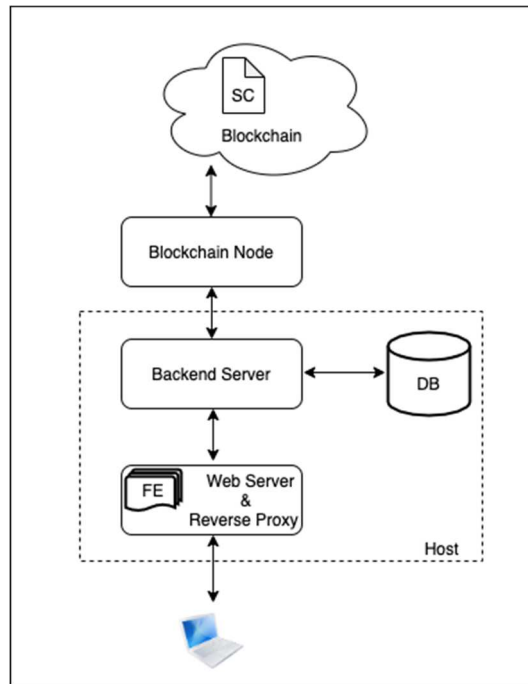


Figure 12 – Architecture diagram

The entry point of the application is the web server, which provides static assets for the frontend and acts as a reverse proxy. The backend server is responsible for holding the business logic and the connection to both the database and blockchain node. The database stores the application data that is mutable (*i.e.* can change over time). The blockchain node connects the application to the decentralized ledger, which has a smart contract with immutable information (*i.e.* do not change over time).

Both backend and frontend were developed with frameworks. A software framework is a software environment that provides generic features, tools, libraries and a pattern to develop a software application. Its goal is to facilitate software development. Typically, it cannot be modified however it can be extended via overriding.

Figure 13 illustrates the application architecture of the POC with its components and corresponding frameworks, libraries and protocols. Each component is explained in the following sections.

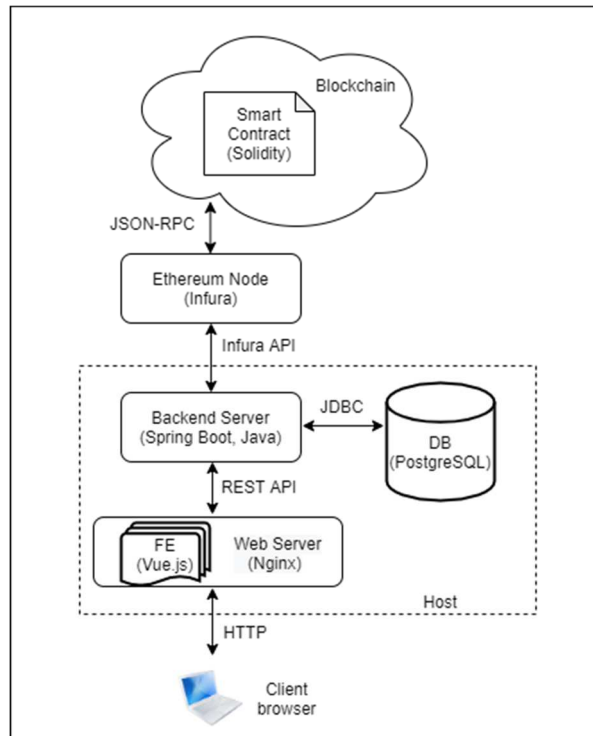


Figure 13 – Application architecture

3.4.3 Web server

The web server is responsible to handle and process client requests through HTTP (Hypertext Transfer Protocol). It also stores the static assets of the frontend application (Hypertext Markup Language - HTML pages, Cascading Style Sheets (CSS), JavaScript files, fonts and images) and acts as a reverse proxy to forward the requests to the backend Application Programming Interface (API) (Mozilla, 2019).

Nginx was the chosen software for the web server because it can be both a web server and a reverse proxy. It is open source and distributed under the 2-clause BSD-like license (Nginx, 2020).

Each client request is handled by Nginx. If the request is done to the root (“/”) then the web server replies with the frontend assets, otherwise the request is forwarded to the backend server REST API by the reverse proxy, preventing the backend server from being exposed to the internet and being only accessible internally. REST (Representational State Transfer) is a software architectural style defined by standards. REST allows a variety of data formats and has a good performance (Fielding, 2000).

3.4.4 Frontend

The frontend application is a Single-Page Application (SPA), which allows to update only the necessary content of a web page instead of loading new pages from the server. The page content is updated through requests to the backend API. SPAs provide a better user experience and higher performance (Mozilla, 2020).

Vue.js was the SPA framework used to develop the user interface (version 2.6.10) (Evan, 2019). This framework was chosen due to its small size (a full-featured application compressed has around 30KB), ease of use, flexibility and detailed documentation. The framework is open source with an MIT license.

Vue.js applications are composed of a set of components. A component is a small, self-contained and reusable software. A typical component is divided into three sections: template, script and style.

The template section holds HTML-based syntax that is parsed by compliant browsers and allows to bind the data of the Vue instance to the rendered Document Object Model (DOM). The script section aggregates the JavaScript code of the component, having a specific structure to hold local component data, methods and lifecycle hooks. The style section holds the CSS style for the component.

The template section of Vue.js components is compiled into a Virtual DOM render functions, allowing to re-render the minimum changes in the DOM when the state of the application changes.

A component library was used to improve the development of the User Interface (UI) components. The chosen library was BootstrapVue (BootstrapVue, 2020), a wrapper of the popular UI library Bootstrap, developed by Twitter, specific for Vue.js and distributed under an MIT license.

The frontend was developed with the support of external dependencies (packages) that were added from Node Package Manager (NPM) (*e.g.* BootstrapVue). The compilation and bundling of the frontend components and their dependencies were done through Webpack (Webpack, 2020) into static assets that were deployed in the Nginx web server. These assets are downloaded by the clients when the first request to the web server is performed and interpreted by the browser.

3.4.5 Backend server

The backend server manages the business logic of the application. It was implemented in Java (Java Development Kit version 11) (Java, 2019), because it is robust, easy to use, platform-independent and secure (IBM, 2019). Another advantage of Java is the existence of several frameworks (*e.g.* Spring or Java Server Faces).

Spring Boot was the chosen framework (version 2.1.6) (Pivotal, 2019). It is an extension of the Spring framework that eliminates the need for some Spring configurations, has an embedded server that reduces deployment complexity and is an open source project distributed under an Apache 2.0 license (Baeldung, 2019).

The backend server is divided into three layers: web, service and repository (see Figure 14). It allows following some of the software development best practices: separation of concerns between layers, code reusability and simplicity.

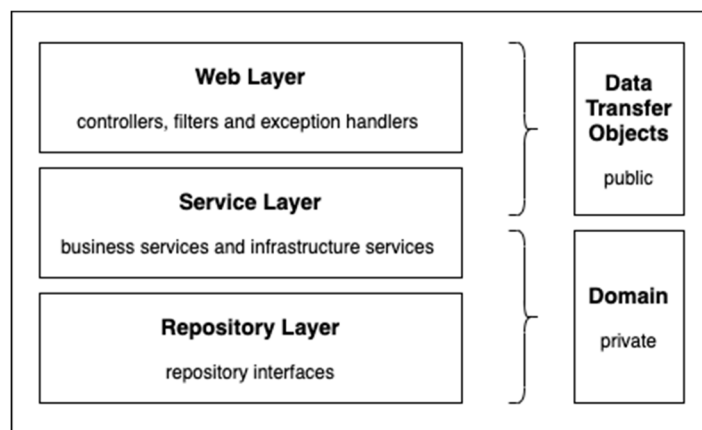


Figure 14 – Backend server layers

The web layer is the entry point of the server and is responsible for processing and validating the user input, handling authentication and protecting the API from unauthorized users. It has controllers, filters, exception handlers and exposes the REST endpoints. The JSON Web Token (JWT) (Peyrott, 2016) is the method used to represent claims between the backend and frontend in a secure way.

The service layer has both business and infrastructure services. The business service is responsible for encapsulating all business logic and handling the repository interactions, while the infrastructure service handles the communication between the application and the Ethereum node.

The repository layer acts as a data access abstraction and is responsible for the interaction between the application and the database through the Java Database Connectivity (JDBC) API. The repository uses the Java Persistence API (JPA) specification for data persistence and conversion from a Java object into a database entity (and vice-versa). The used JPA implementation was Hibernate with Spring Data JPA library.

The domain model is used as an entity object. The communication between layers is performed by a Data Transfer Object (DTO), which is an object used to carry data between web and service layers and to expose the data to the API consumers. This enforces extra security as it avoids the exposition of the domain model.

All actions that store information in the database and smart contracts are performed synchronously to avoid inconsistent information between the database and the smart contract. In case an error occurs in the database or blockchain transaction, both are rolled back.

3.4.6 Database

Since the information stored in a public blockchain is public, the sensitive information is stored in the database. There is also persisted the mutable data of the application.

A relational database was chosen as the stored data has a strong relationship with each other. The selected database was PostgreSQL (PostgreSQL, 2019). It is an open source object-relational database system distributed under PostgreSQL License, a liberal license similar to the MIT license.

For development purposes an in-memory H2 Database Engine (H2, 2019) running in PostgreSQL compatibility mode was chosen as it allows lower configurations and enables faster development. It is open source with a dual license: Mozilla Public License version 2.0 and Eclipse Public License.

The data model is presented in Figure 15. All entities have a basic auditing mechanism by recording when was the data created and last updated.

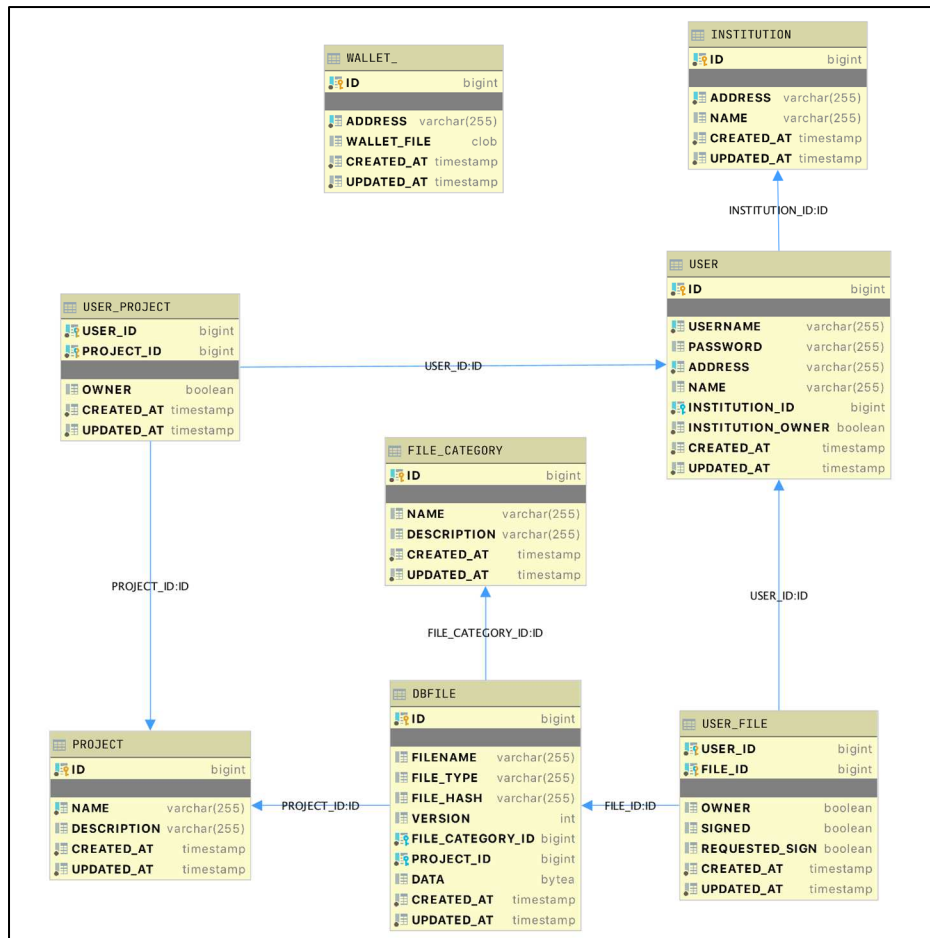


Figure 15 – Database entity-relationship diagram

Each time an Institution entity is added a User entry is created with the INSTITUTION_OWNER flag set to true. This user is the admin user of the institution. During the institution and admin user creation, a new Ethereum wallet is generated and stored in the Wallet entity. Both entities will have the same Ethereum address, derived from the Ethereum wallet.

A User entity can have multiple files (called DBFile to distinguish between native Java File class) and a file can be available to multiple users (many-to-many relationship), so a join table (USER_FILE) was created with a composite primary key formed by USER_ID and FILE_ID. Some attributes of the File entity are the filename, file type (the content type of the file) and the version, which allows the users to upload new versions of the file. Each file belongs to a certain category, that can be Technical Drawings, Contracts or BIM files, and to a certain project.

The USER_FILE join table also has as Boolean attributes the OWNER, SIGNED and REQUEST_SIGN. The first indicates if the user is the owner of the file (*i.e.* if the user is

the author of the specific file version), the second shows if the user signed that specific version of the file and the last one indicates if the user is requested to sign a specific version of the file.

Each time a file is shared, or a signature is requested, a new entry in `USER_FILE` is created, this allows to only show to the users the files they have access to.

A user can be assigned to multiple projects and each project can have multiple users (many-to-many relationship), so a join table was needed: `USER_PROJECT` with a composite primary key formed by `USER_ID` and `PROJECT_ID`. This join table also indicates if the user is the owner of the project. Only project owners can add users to the project or delete the project itself. Typically, the project owner is also the institution owner.

To be compliant with General Data Protection Regulation (GDPR) privacy policies, the User and Institution entities do not store private information besides username and name, which can also be replaced by some other identification within the company.

3.4.7 Blockchain

Ethereum was the chosen blockchain implementation as it allows the development and deployment of smart contracts.

All blockchain interactions are performed with the central authority wallet. This avoids the need of each institution and its users to have the knowledge of blockchain concepts, as Ether acquaintance or *gas* price. For this reason, the connection between the application and the Ethereum network is performed on the server side, in the backend server.

The library chosen to interact with the Ethereum client was Web3j (Web3j, 2019), which provides an implementation of the JSON-RPC client API in Java. JSON-RPC is a remote procedure call (RPC) protocol, that defines data structures and rules for their processing. It can be used over HTTP and uses JavaScript Object Notation (JSON) (JSON, 2019) as the data format. JSON is a data-interchange format easy for human reading and machine parsing, that can represent strings, numbers, ordered sequences of values and collections of name/value pairs (Ethereum, 2019b).

To avoid the need of having a local node client running with an entire blockchain, the Infura Ethereum node cluster as a service was chosen (Infura, 2019). The Ropsten testnet

network was used in the POC during the development and test of the application to avoid spending money during transactions' execution.

3.4.8 Smart contract

Solidity was the selected programming language for the development of the smart contract as it is currently considered a standard, has detailed documentation and is the most supported and maintained language (Ethereum, 2019c).

The conversion from Solidity high-level contract source code into bytecode, the format that is interpreted by the EVM, is performed by the solc compiler.

Remix (Remix, 2019) was the Integrated Development Environment (IDE) used, which is a browser-based IDE and compiler that allows the development of Ethereum contracts with Solidity language and transaction debugging.

The smart contract, deployed in Ropsten network, records the information that is intended to be immutable and tamper-proof. In the current POC was decided to store in the blockchain the following data:

- Institution - name and its own Ethereum address;
- User - name, its own Ethereum address and the Ethereum address of his institution;
- File - hash, the Ethereum address of its owner, filename, creation data and Ethereum address lists of the requested signatures and signers.

It was decided to store the minimum information on the smart contract as possible due to GDPR privacy policies. To have a reference in common with the database, an Ethereum wallet is generated through the Web3j library for each new institution and user creation. The wallet file is stored on the database, while the address is used to link the user to the smart contract.

The wallet file stored on the database allows creating an abstraction of the blockchain interaction to the user. It can also be used in the future modules of the application, such as automatic payments.

Due to security concerns, an additional variable was added to the smart contract to prevent unauthorised access to it - applicationOwner. The variable is assigned during deploy time to the Ethereum address of the central authority wallet.

To assure that the smart contract state can only be changed by a transaction sent from the central authority, all functions that change the smart contract state are protected to check if the address of the transaction sender is allowed to change smart contract data, *i.e.* if the transaction is signed by the central authority address. This prevents the smart contract state to be changed by transactions that are not sent by the application.

The smart contract has two types of functions: read and write. The write functions are the ones that change smart contract state and are used to register a new file, user or institution on the blockchain, or to modify some of their attributes, *e.g.* add a signer or a signature request to an existent file. They are intended to be only used by the application, so their execution is protected and restricted to only execute if the transaction sender is the Ethereum address of the central authority (`applicationOwner`) (see Figure 16).

```
function addFile(bytes32 _fileHash, address _owner, bytes32 _filename) public {
    require(msg.sender == applicationOwner, "Sender not allowed to add file");
    File storage file = files[_fileHash];
    require(file.fileHash[0] == 0, "File already exists");

    User memory user = users[_owner];
    require(user.addr != address(0), "User does not exist");

    file.fileHash = _fileHash;
    file.owner = _owner;
    file.filename = _filename;
    file.creationDate = now;

    allFiles.push(file);
}
```

Figure 16 – Smart contract function to register a file

The read functions are the ones designed to be publicly used to retrieve the smart contract state. These functions are used to get file information (*e.g.* file signers and signature requests) and user information (*e.g.* name and institution). These functions are important to be kept public because they will be used to verify if a file is registered on the blockchain and who is the file owner, the file signers, the pending signatures, and their respective institutions.

To protect the data on the smart contract the *getFile* function (see Figure 17) can only be called by having the file hash, this replicates the real-world behaviour, where a file

authenticity and their signatures can only be verified by having the file itself. Similarly, the function to get the user info can only be invoked by knowing his Ethereum address.

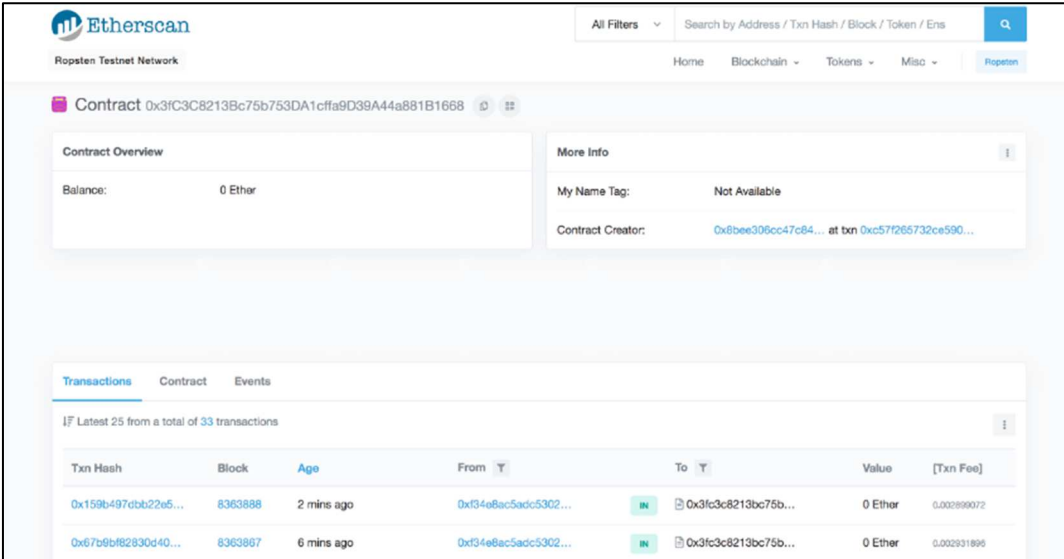
```
function getFile(bytes32 _fileHash) public view returns (bytes32 fileHash,
    address owner, bytes32 filename, address[] memory signatures,
    address[] memory signRequests, uint creationDate) {

    File memory file = files[_fileHash];
    fileHash = file.fileHash;
    owner = file.owner;
    filename = file.filename;
    signatures = file.signatures;
    signRequests = file.signRequests;
    creationDate = file.creationDate;
}
```

Figure 17 – Smart contract function to retrieve file information

The current application can be used to verify the authenticity of a file by uploading a file to it. It generates a file hash and retrieves its information from the blockchain. However, in case the application no longer exists, a call directly to the smart contract or by another frontend application can be done to evaluate the file authenticity. This assures that the decentralized information will always be available and is independent of the application that registered it.

The interactions on the smart contract can be visualized on Etherscan in Ropsten Network (Etherscan, 2020). An example of the contract interactions is presented in Figure 18 and the transaction detail in Figure 19.



The screenshot shows the Etherscan interface for a smart contract on the Ropsten Testnet Network. The contract address is 0x3fC3C8213Bc75b753DA1cfa9D39A44a881B1668. The contract overview shows a balance of 0 Ether. The 'More Info' section indicates that the name tag is not available and shows the contract creator as 0x8bee306cc47c84... at transaction 0xc57f265732ce590... Below this, the 'Transactions' tab is active, displaying a table of the latest 25 transactions out of a total of 33.

Txn Hash	Block	Age	From	To	Value	[Txn Fee]
0x159b497cbb22e5...	8363888	2 mins ago	0x34e8ac5acc5302...	0x3fC3c8213bc75b...	0 Ether	0.002890072
0x67b9b82830d40...	8363887	6 mins ago	0x34e8ac5acc5302...	0x3fC3c8213bc75b...	0 Ether	0.002931896

Figure 18 – Smart contract transactions

Each interaction is detailed in Figure 20 with the corresponding interaction between the architecture components.

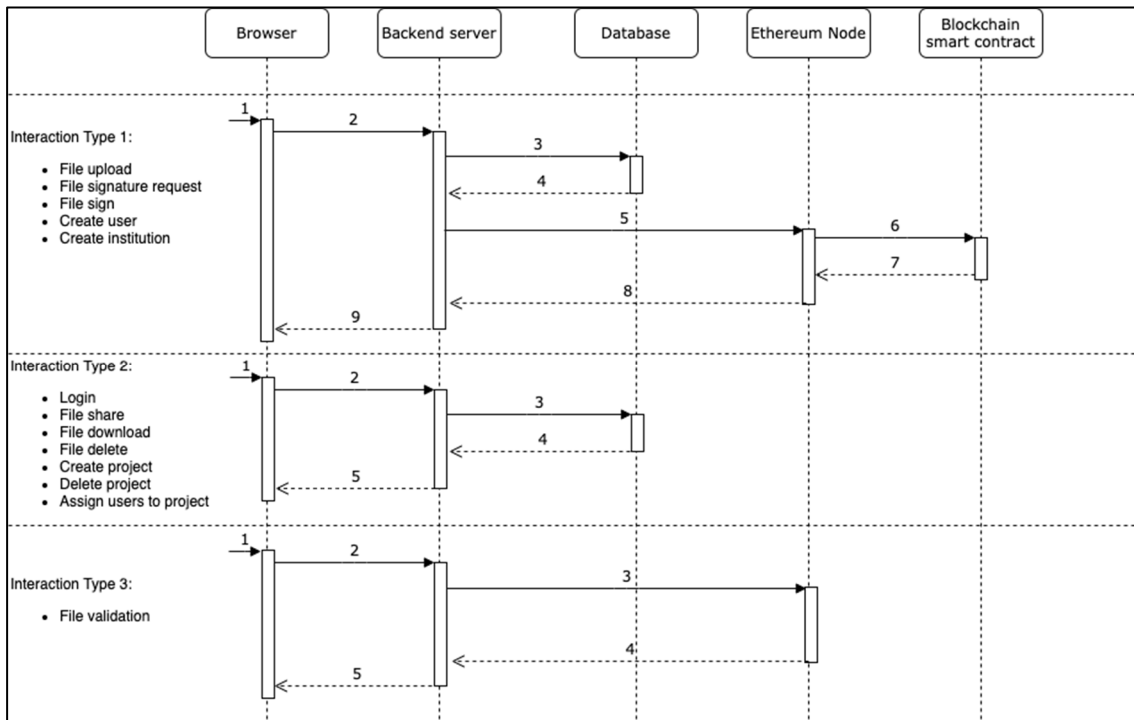


Figure 20 – Architecture components interplay for each user interaction

Interactions type 1, with file upload as an example:

1. A user accesses the application in a browser and uploads a file;
2. The frontend application, running in the user's browser, performs a backend server request – HTTP post request;
3. The backend server generates the hash of the file;
4. The file information is persisted on the database, including the file itself;
5. The backend server submits a transaction to the Ethereum node signed by the Central Authority if no database error is thrown;
6. A transaction is submitted to the Ethereum network;
7. The transaction is executed by the Ethereum network;
8. The transaction result is returned to the backend server. In case an error occurs the database transaction is rolled back;
9. The operation result is returned to the browser warning the user about its success.

Interactions type 2, with file share as an example:

1. A user accesses the browser and shares a file with another user;
2. The frontend application performs a backend server request;
3. The backend server updates the users that have access to the file;
4. The information is persisted on the database;
5. The operation result is returned to the browser warning the user about its success.

Interaction type 3, with file validation:

1. A user accesses the browser and uploads a file to be verified;
2. The frontend application performs a backend server request;
3. A query to the Ethereum node is performed with the file hash;
4. The Ethereum node returns the information about the file;
5. The details of the file are returned to the browser.

3.5.1 Institution and admin user creation

Institutions are created by the central authority, the entity responsible for the application maintenance and verification of the client's identity, assuring that a client is who he claims to be. The institution creation is performed through a POST request with an authorisation header with a valid JWT token, containing the identity of the central authority, and a body with the institution name for the institution creation and the name of the user, username and password for the admin user creation. For the current POC the admin user password is defined by the central authority however, this should be improved in its implementation.

The institution creation generates an Ethereum wallet file and creates the admin user of the institution. Both admin user and institution share the same wallet file and, consequently, the same Ethereum address.

During the creation, both institution and admin user data is stored in the database and a transaction is submitted to the blockchain network to create them on the smart contract, storing their name and address. In case an error occurs, both blockchain and database transactions are rolled back.

3.5.2 Login

All users of the application need to perform a login to access the application unless they only want to access the verification page (see Figure 21).

The login process generates a valid JWT token in the backend server based on a secret stored in the backend server with the subject, Ethereum address and expiration time (see Figure 22). The token is attached to all requests by the browser and is responsible to validate the user identity (*i.e.* the user Ethereum address). This leads to a stateless backend server as it does not need to keep a state between user requests. The token expiry date is used for security reasons as it is only valid for a certain period.

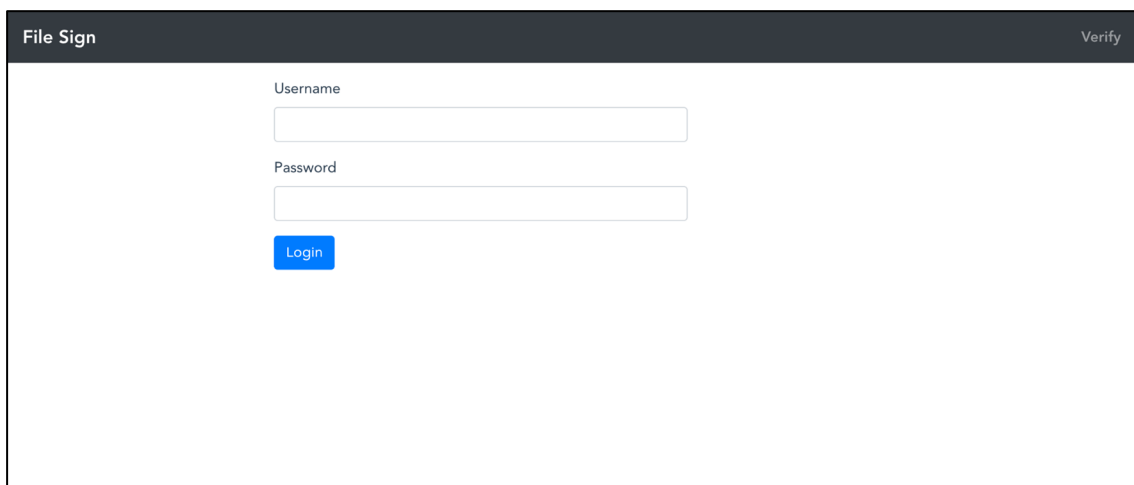


Figure 21 – Login page

```
{
  "sub": "tiagomota",
  "address": "f34e8ac5adc5302082cc685a6ce996a16ea1dfd6",
  "exp": 1594841399
}
```

Figure 22 – Example of a JWT token generated after a successful login

After a successful login, the users are redirected to the main page of the application - the dashboard (see Figure 23). There the users can visualise their projects (1), upload new files (2) and see their previously uploaded files (4) within each category (3), associated with the selected project (1). The user can also access the verification page to verify the authenticity of a specific file (5), see the authenticated user and perform the logout (6).

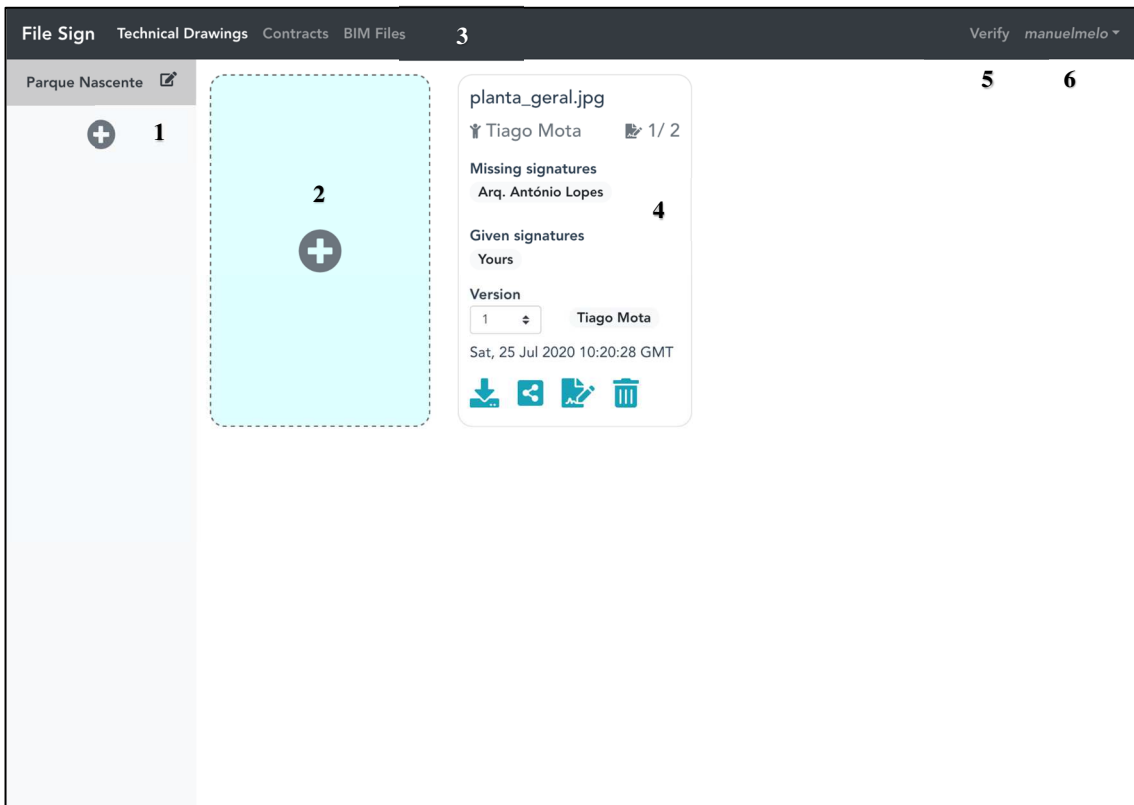


Figure 23 – Application dashboard

3.5.3 User management

The admin user is presented with an empty dashboard in the first application access (see Figure 24). There the user can manage the users of its institution by clicking in “User Management” under the username dropdown – can create new users or delete them (see Figure 25). This menu entry is only available for admin users.



Figure 24 – Empty dashboard

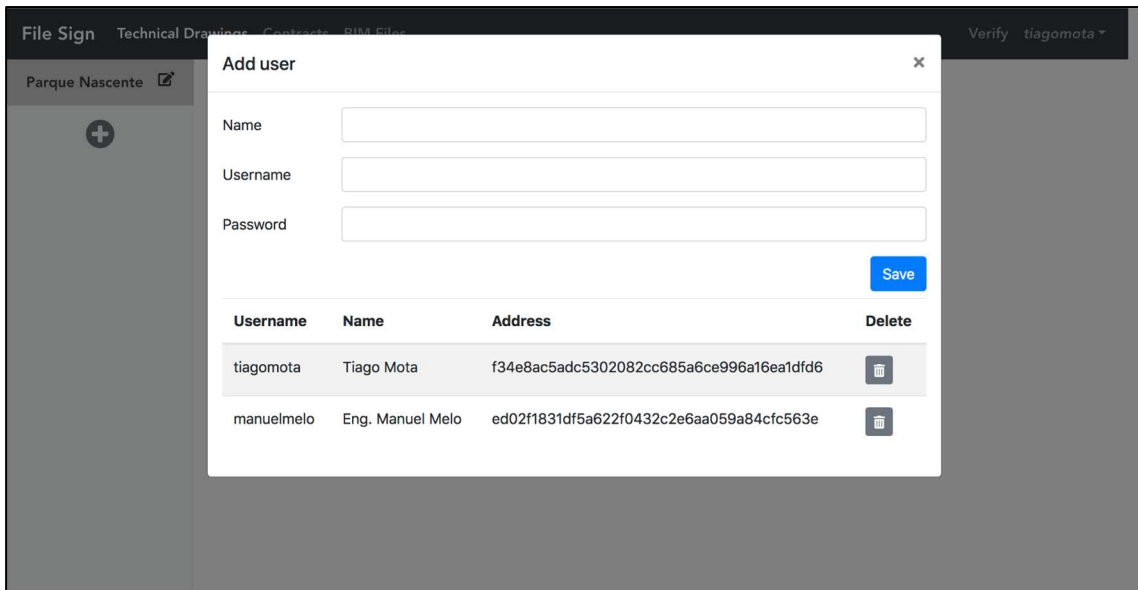


Figure 25 – User management dialog

The application generates an Ethereum wallet for each created user, providing an Ethereum address. This address is the identity of the user within the application and on the blockchain.

The user creation stores the user information in the database and submits an Ethereum transaction to record the user in the smart contract, storing the Ethereum address, name and Ethereum address of the institution.

As in the creation of the admin user, the user's password is defined by the admin user of each institution. However, this should be improved on its implementation.

3.5.4 Project management

Admin users can also manage the projects of their institutions. To create a new project the admin user clicks on the plus button in the project section. This action displays a dialog where the user inserts the project name, description and the users of the institution that can contribute to it (see Figure 26). When the project is created an alert is shown with a message indicating the success of the operation (see Figure 27).

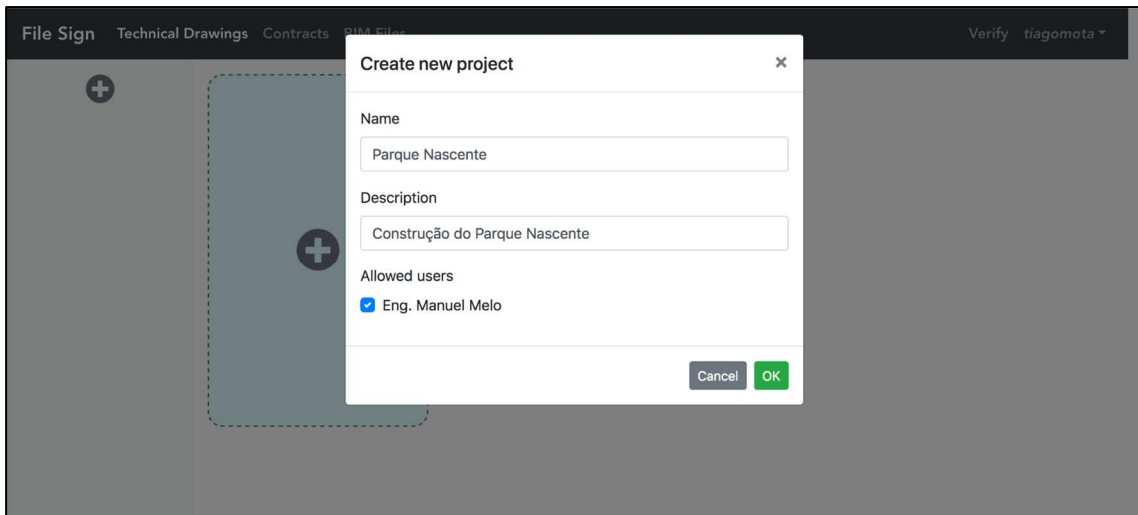


Figure 26 – Project creation dialog

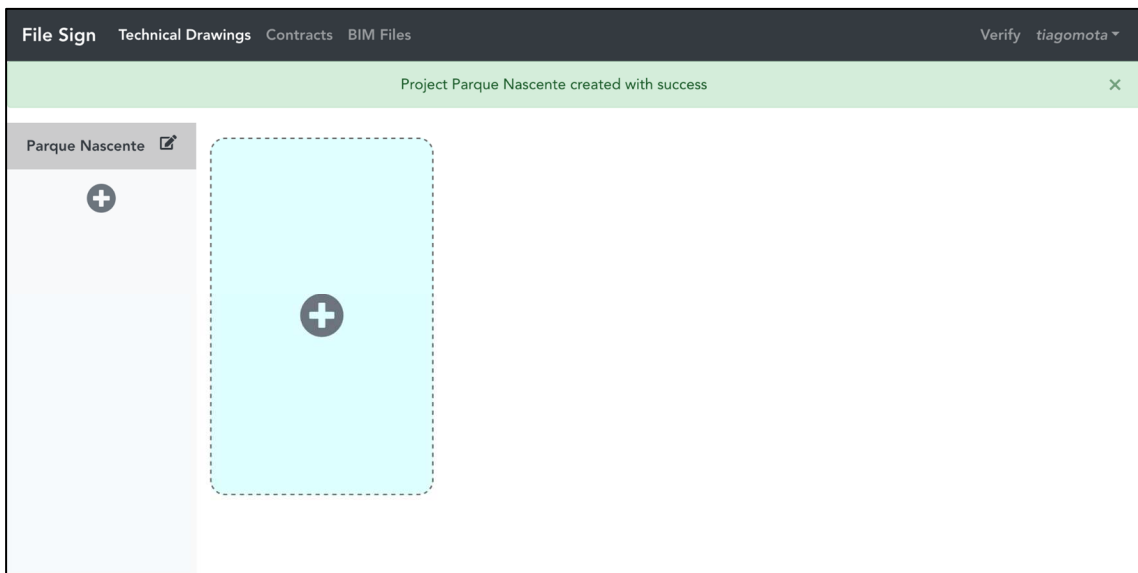


Figure 27 – Alert showing that the project was created with success

It is also possible to edit or delete a project by clicking on the edit button next to the project name, there the admin user can change the name, the description, the users that have access to it or delete the project (see Figure 28).

The project information is only recorded in the database because it is used to organise the files in the application, not being required for the file's integrity verification.

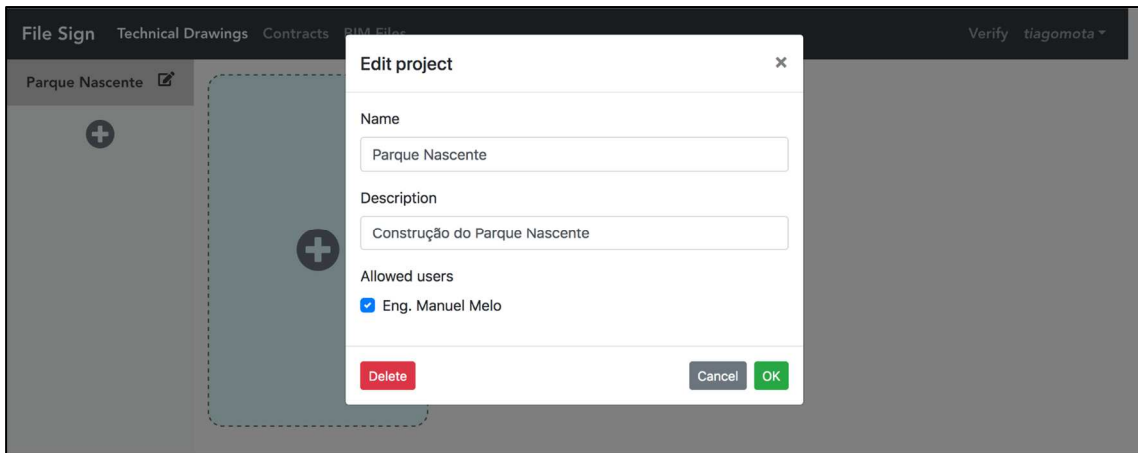


Figure 28 – Project edit or delete

3.5.5 File upload

Each user of the application can upload files into the projects they have access to. A drag-and-drop dashed section is available to add a new file into the selected project. Alternatively, the card can be clicked and the browser dialog will appear to select the file from the filesystem (see Figure 29). Each file that is uploaded to the application is stored in a database and its hash is registered in the smart contract. The hash of the file is generated through the Keccak-256 cryptographic function (Bertoni *et al.*, 2009).

A loading wheel is shown while the file is being processed (see Figure 30). A confirmation alert will be displayed and the file will appear on the dashboard when the upload and smart contract registration is complete (see Figure 31).

The files are grouped by versions. To upload a new version the user needs to upload a file with the name of the original file. Each version has its metadata, such as the file owner, uploaded date, requested and given signatures. This behaviour simulates a signature of a physical file, where each signature is valid only for a specific version of a document.

When a specific version is shared or a signature is requested from another user, he/she will be able to upload a new file version. Each version is incremented automatically and sequentially.

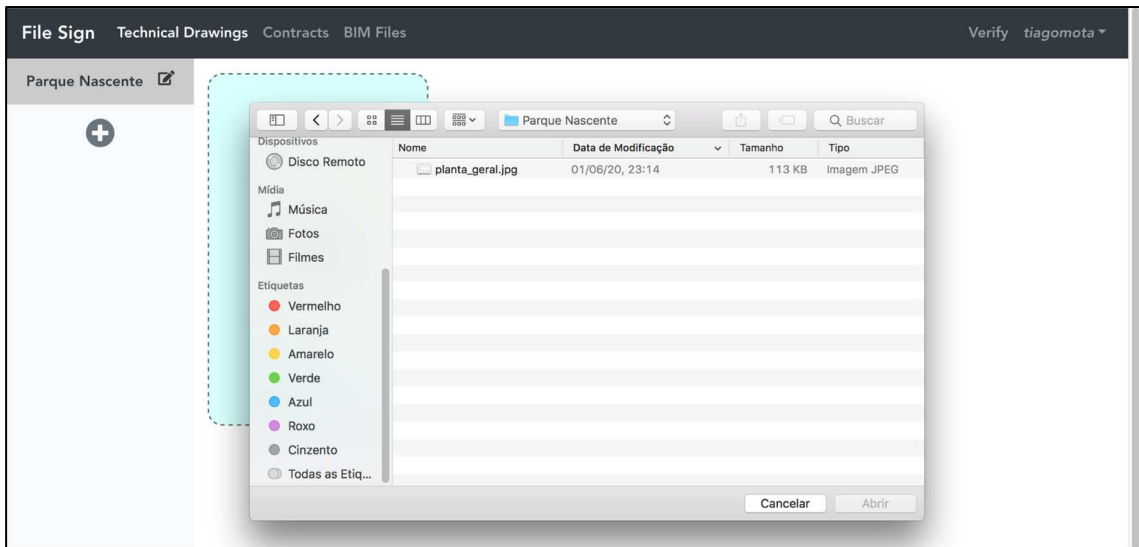


Figure 29 – File upload dialog



Figure 30 – File upload processing

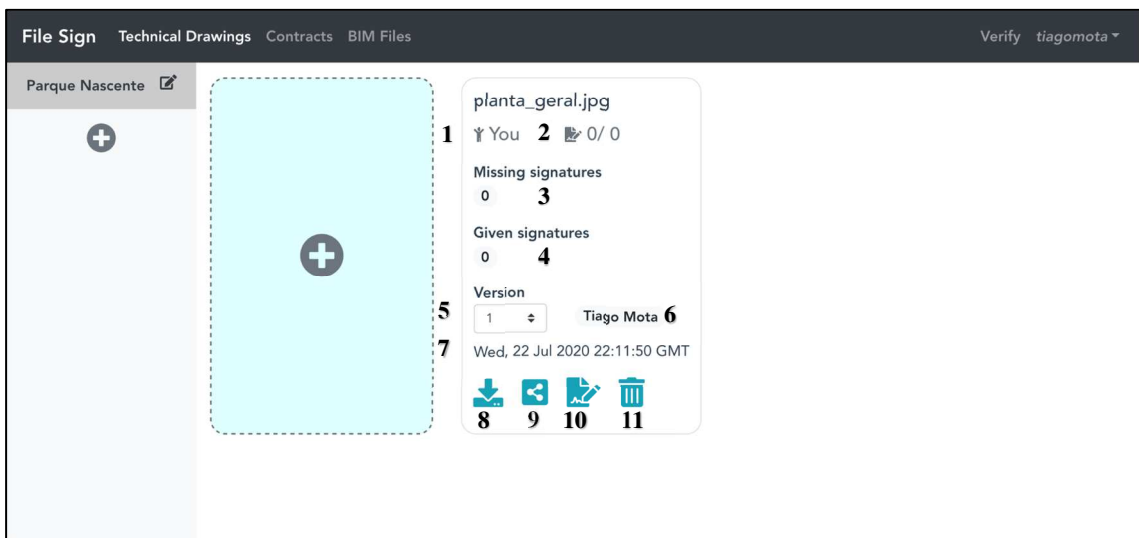


Figure 31 – Dashboard with an uploaded file

The uploaded file is only available to the user that uploaded it until he/she shares it with another user or requests a signature. Figure 31 shows the card with the uploaded file, where the information related to the selected file version is displayed through icons:

1. File owner;
2. Total of signatures given / total of requested signatures;
3. Identity of the users that have pending signatures;
4. Identity of the users that have signed the file;
5. File version;
6. Author of the current version;
7. Registration date of the file version;
8. Download button;
9. Share button;
10. Request signature button;
11. Delete button.

To visualize the information from another version or perform any action on it, the user needs to select the desired version from the version dropdown (5).

3.5.6 File share

The file share gives the possibility to share a file without having the need to request a signature and sign a document. Consequently, this interaction is only persisted in the database.

Users can share their files by clicking on the share button. The button triggers a dialog to select with which users of the project they want to share the file (see Figure 32).

Once a file is shared, the users that receive access to the file can interact with it by downloading, sharing it with other users, requesting signatures or uploading a new version. The users will also visualise the new file versions uploaded by any collaborator.

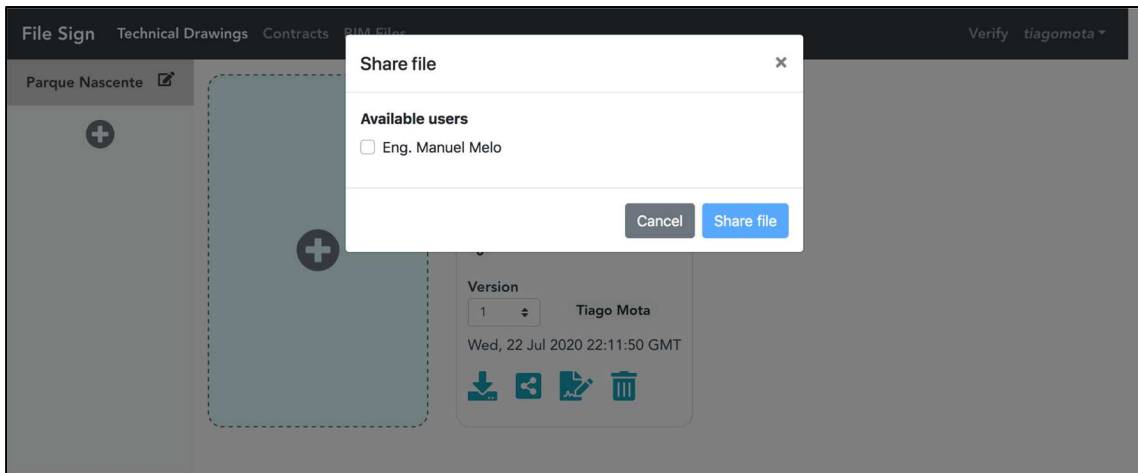


Figure 32 – Share file dialog

3.5.7 File signature request

Instead of simply sharing a file, a user might want to request a signature from other users, meaning that they acknowledge the file content. The user clicks on the request signature button and a dialog with a list of users of the project is shown (see Figure 33). The user selects the collaborators to request the signature and clicks on the “Request Sign” button.

During the process of the signature request a loading dialog is shown since it involves a blockchain transaction that can take some time (see Figure 34). When the processing is complete the file is shown with the name of the user that has a pending signature (see Figure 35).

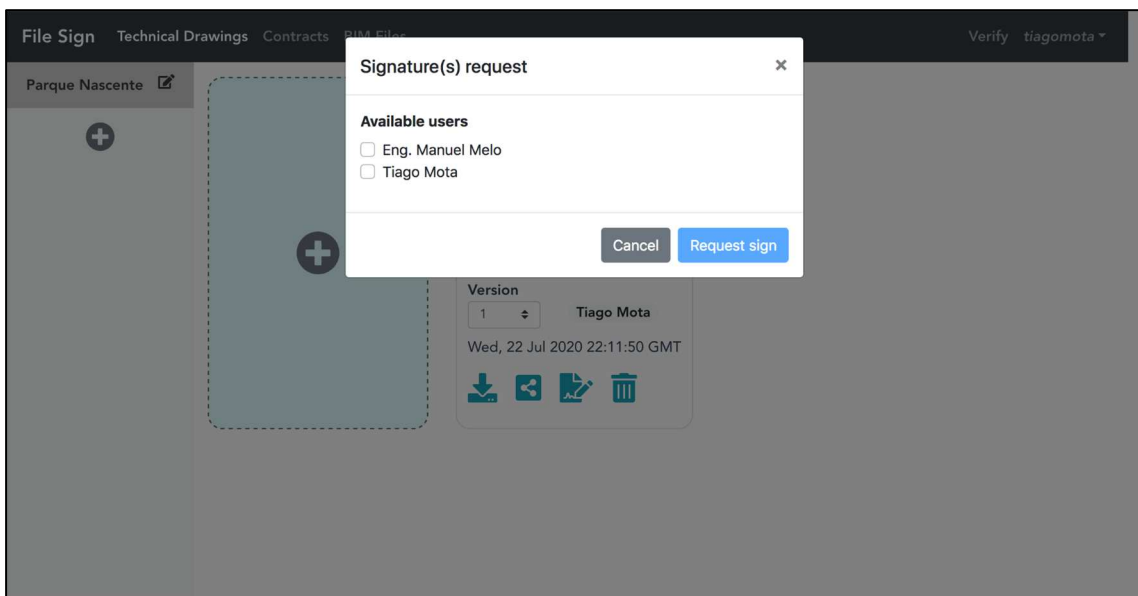


Figure 33 – Request signatures dialog

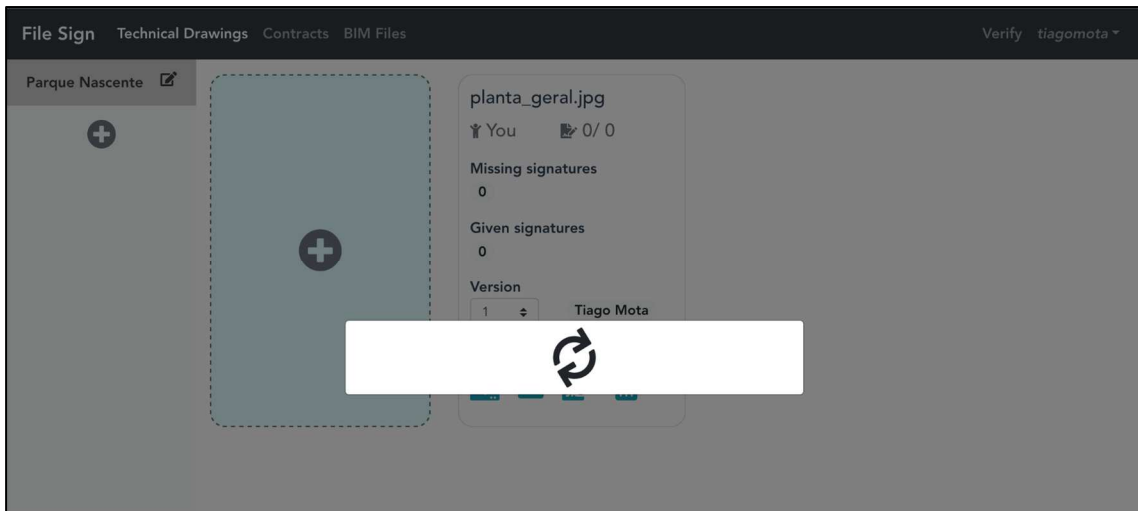


Figure 34 – Request signature processing

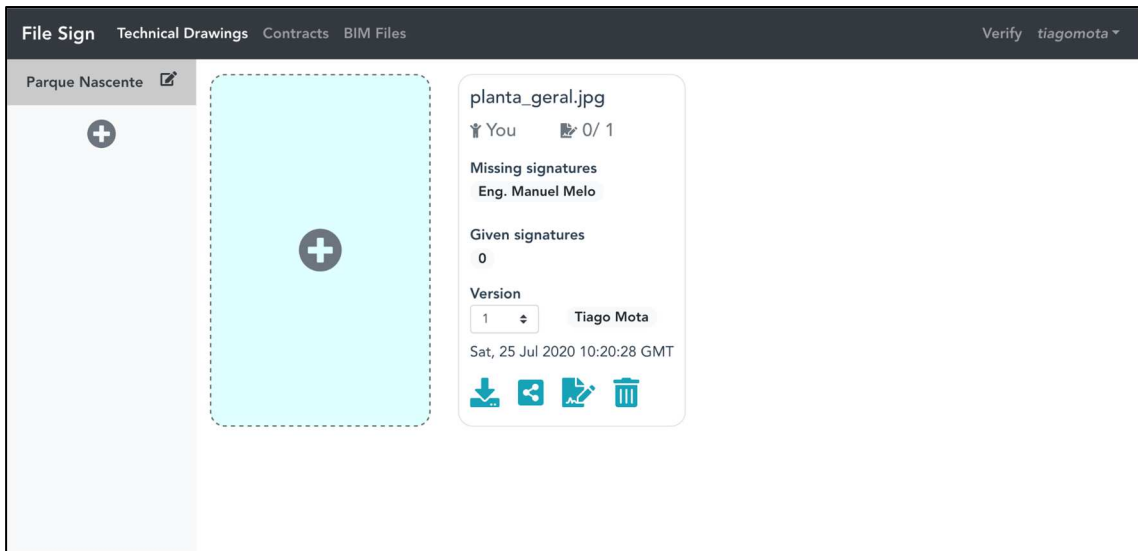


Figure 35 – File info with a pending signature

In the signature request dialog (Figure 33) is possible to visualize that the logged-in user appears on the list. This possibility has been added to allow the current user to request a signature of himself to give its signature later.

The signature request is recorded on both database and blockchain by persisting the Ethereum address of the user who was requested to sign. This information allows everyone that has access to the file to see the pending signatures of the file.

Similarly, as in file sharing, when a signature request is made the user that was requested to sign the file can interact with it by downloading, sharing with other users, requesting signatures or uploading a new version. The same user also has access to the newer versions of the file.

3.5.8 File signing

The missing signatures section shows the name of the users that have pending signatures (see Figure 35). When the user requested to sign the file accesses the application, he visualizes the same section with a “Yours” label followed by a “Sign” button that allows the user to sign it (see Figure 36).

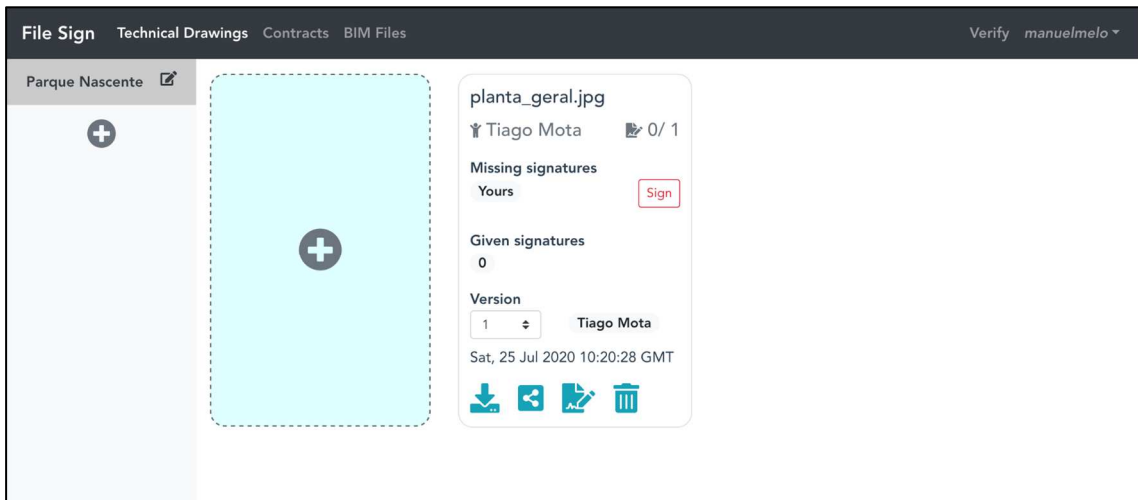


Figure 36 – Pending signature with the label

The click on the sign button triggers an update in the database and a submission of an Ethereum transaction to persist the user signature. The loading wheel is shown until the blockchain transaction is performed (see Figure 37).

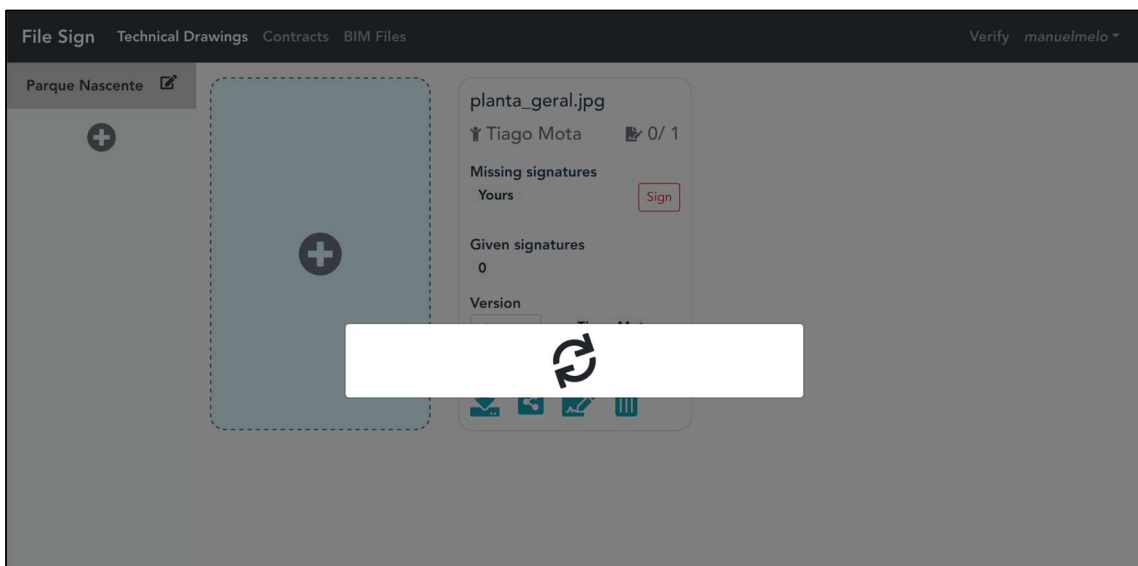


Figure 37 – Registration of a file signature processing

The file info displays the user that signed the file when the signature is registered with success. If the logged-in user is the user that signed the file a “Yours” label is shown (see Figure 38), otherwise the name of the user that signed the file is shown (see Figure 39).

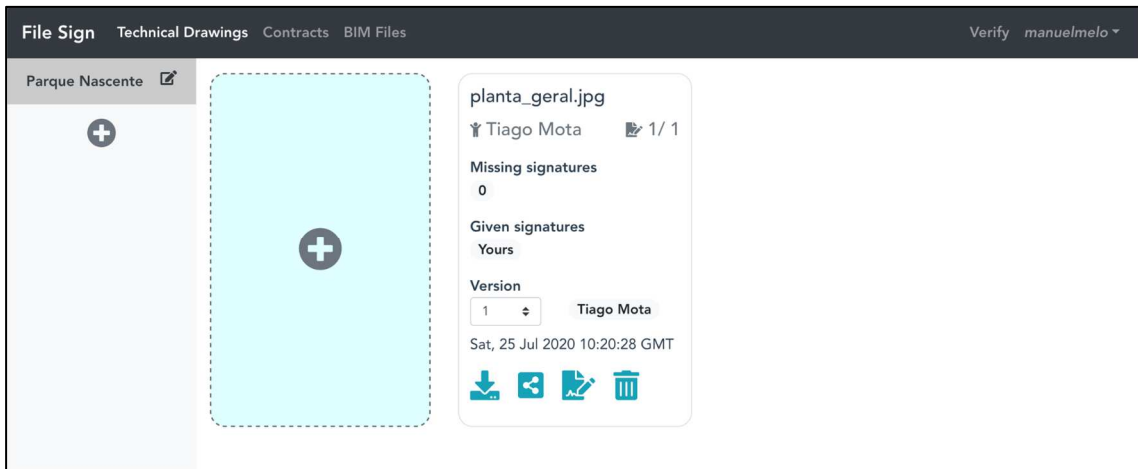


Figure 38 – Given signature with the label

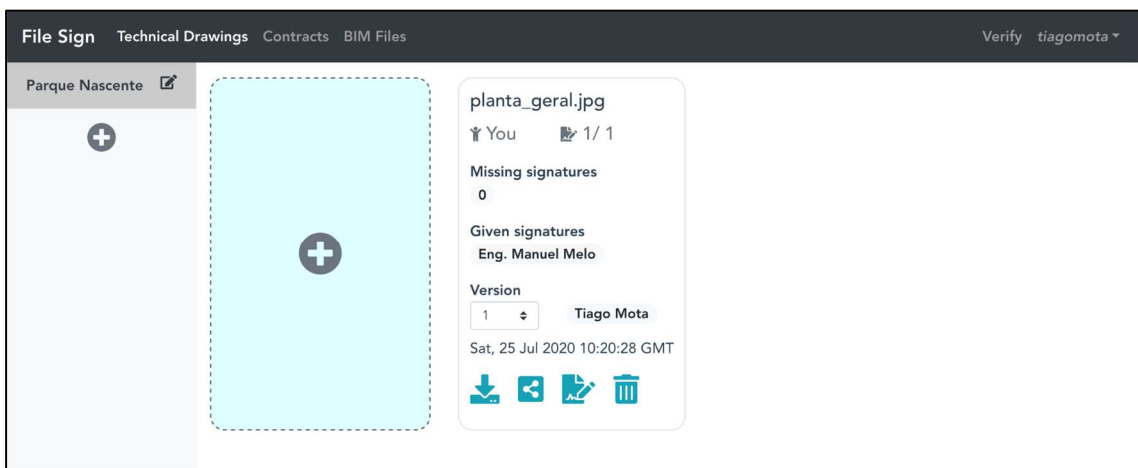


Figure 39 – Given signature with the name of the user

3.5.9 File download

To download a specific file version, the user needs to select the desired version of the file and click on the download button. The download will fetch the file from the database and show a browser dialog to specify if the user wants to open or save the file (see Figure 40).

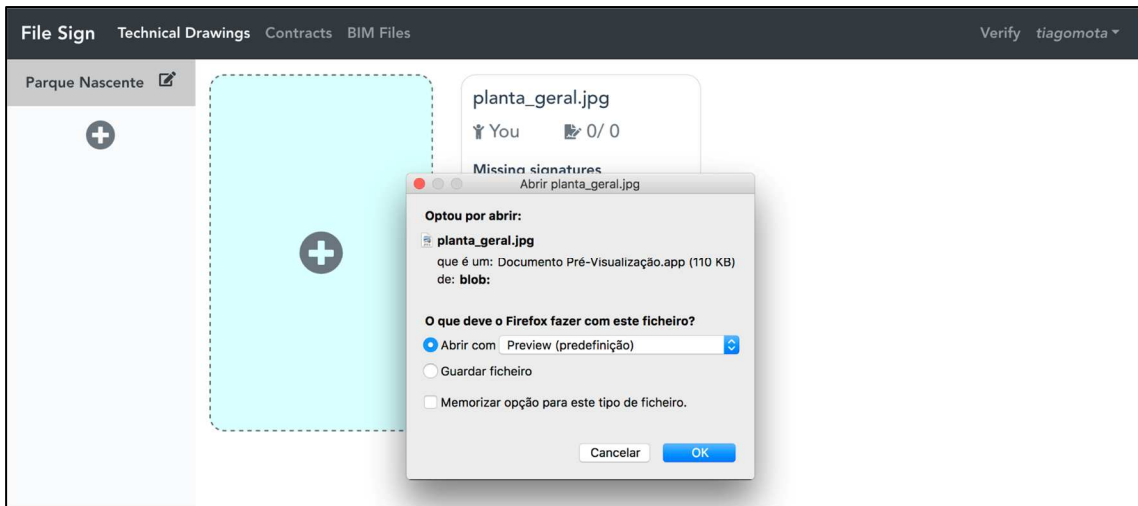


Figure 40 – Download of a specific file version

3.5.10 File deleting

Similarly, to delete a file the user needs to select the desired file version and click on the delete button. It triggers a confirmation dialog (see Figure 41). The delete is only performed at the database and remains on the blockchain. This behaviour is important to keep a track of the file and the possibility to validate it later.

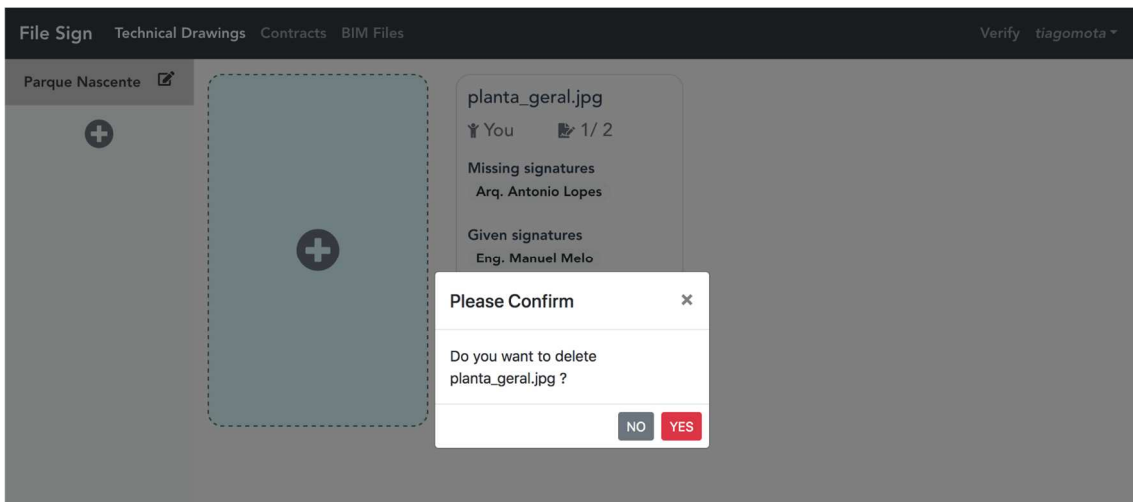


Figure 41 – Confirmation dialog to delete a specific file version

3.5.11 File authenticity verification

The authenticity of the files can be easily verified through a dedicated page on the application (see Figure 42). The verification page can be accessed with and without a login account to allow anyone to verify if a file is authentic.

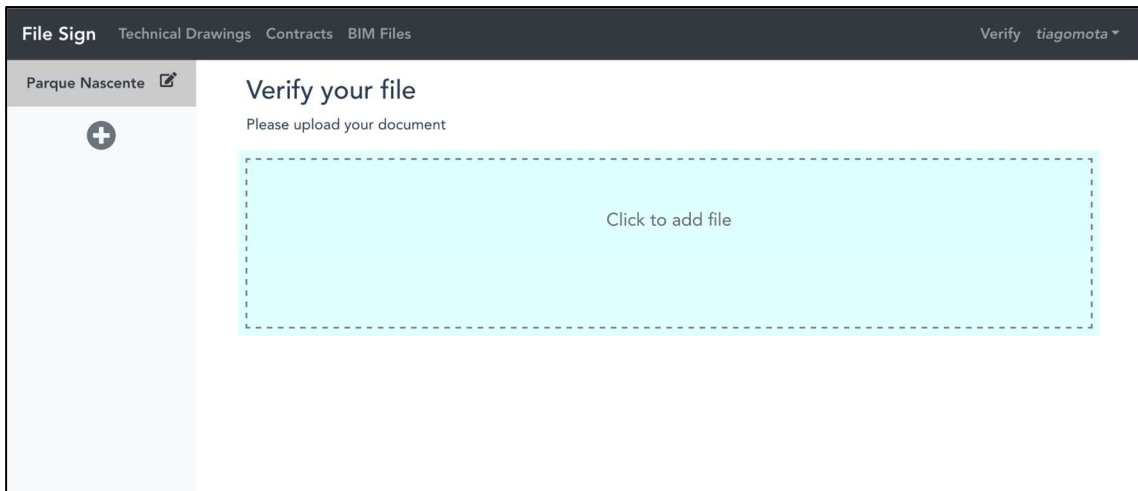


Figure 42 – Verification page

The verification of a file can be done by dropping a file into the dashed area or clicking on this area and uploading the file manually. Uploading status information is shown while the verification is being processed (see Figure 43). During this process, the cryptographic hash of the uploaded file is generated and the Ethereum node queried to get the file information for that hash. If the file exists, all the relevant file information is shown, such as the owner, institution, signers, requested signatures and created date (Figure 44), if not a message saying that the file is not registered is shown (see Figure 45).

Should be noticed that the Ethereum node is queried directly to read information from the smart contract, without a need to submit a transaction to the network, because the node has a copy of the blockchain.

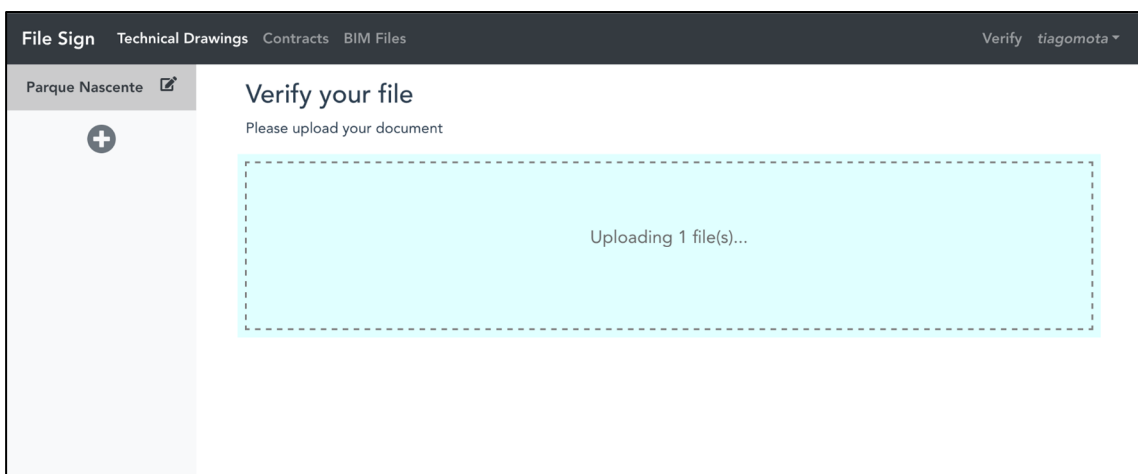


Figure 43 – File uploading for verification

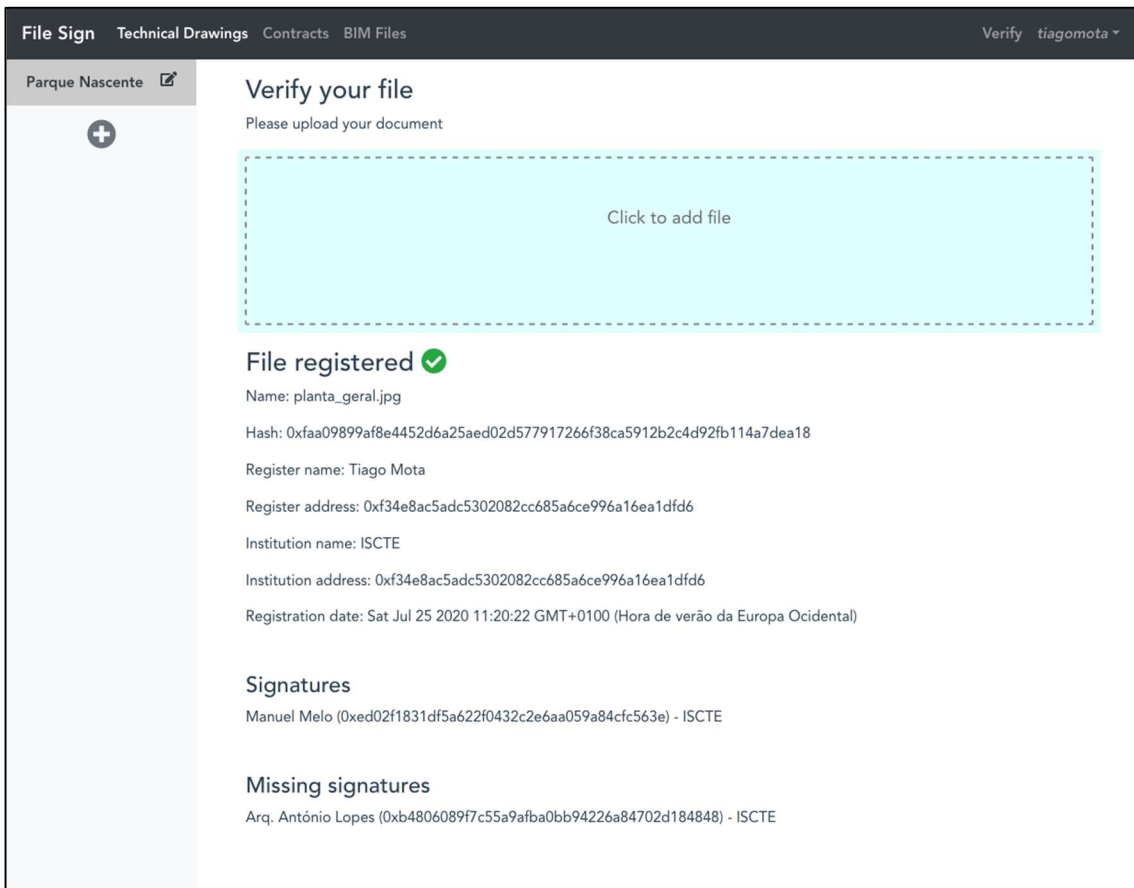


Figure 44 – File verification details

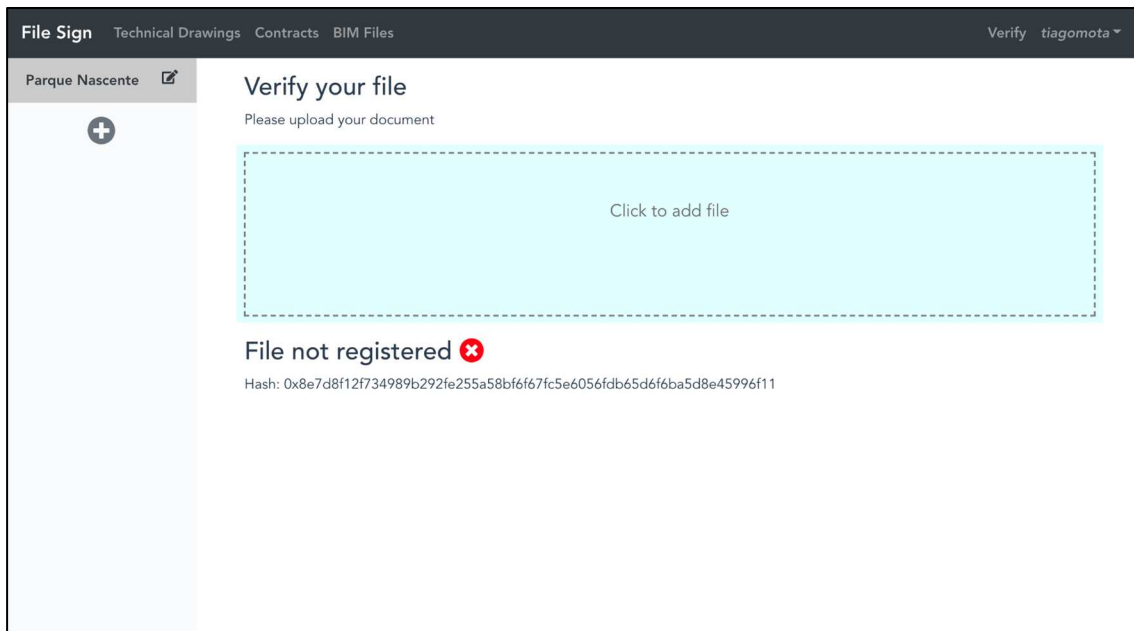


Figure 45 – Verification of an unregistered file

3.6 Summary

The current chapter presented the concept for the information sharing platform in the construction industry. The goal of the platform is to support construction management by facilitating information sharing between all involved participants in all construction phases.

The platform allows sharing files with decentralized records of file modifications and signatures that later can be used as an integrity verification mechanism.

Each entity, company or consortium is represented in the platform as an institution. Their identity is verified by a central authority, which is responsible to create them in the system. Each institution has an admin user that is responsible to create its users and projects within their institution.

A project has files grouped by three categories: Technical Drawings, Contracts and BIM Files. Each file has the version history with the changes performed in the file, containing a record of the date and user that uploaded the file and its versions, and a list of pending and given signatures for each file version.

The POC developed for the conceptual modal was presented, describing all components of the solution architecture and the libraries, frameworks and programming tools chosen for the several layers of the architecture.

Ethereum was the selected blockchain network, the smart contract was written in Solidity and deployed in Ropsten testnet network. Infura Ethereum node cluster as a service was used for the Ethereum node and Web3j as a communication library between the backend server and the network.

The main user interactions with the application are: upload a file, upload a new file version, request another user's signature, sign a file with a pending signature request, share a file, download a file and verify a file. The latest can be performed without an account and is used to see the file information recorded on the blockchain.

Chapter 4 - Results and discussion

4.1 Introduction

The current chapter aims to present a discussion about the conceptual model and how it answers the research question. It also evaluates the performance of the POC considering the transaction duration and cost to execute the smart contract function responsible to register a file.

4.2 Conceptual model

The proposed platform shows that blockchain technology can bring value to the construction industry because it can be used as a mechanism of trust between construction participants.

Although blockchain allows the decentralization of information, the central authority concept is needed to maintain the application and validate institutions' identity during their registration within the platform. This identity validation is important since the information will be stored in an immutable and decentralized ledger that can later be used as an integrity verification mechanism.

An important characteristic of the conceptual model is the separation of concerns between the central authority and institutions. The central authority does not interfere with the institutions, its responsibility is constrained to its creation. However, this comes with a drawback of the central authority not being able to validate the institution users' identity, having to trust in the admin users of the institutions.

It is crucial to have the files and their information registered on a public blockchain network to enable anyone with access to a file to validate it. This allows anyone to verify if a file is registered on the blockchain, their pending signatures and who has signed the file. This also assures that even the platform no longer exists, the information recorded on the blockchain will remain forever. However, only those who have a copy of the file can check the document information.

Another important concept is the abstraction of blockchain knowledge from the platform's users as they do not need to understand the underlying technology to use it. This leads to the usage of the central authority cryptocurrency wallet to submit blockchain transactions and the need to set up a business model to support the expenses of the information registered on the blockchain.

Theoretically, the time that takes to process a transaction by the Ethereum network depends on the *gas* fee associated with the transaction, with faster transactions being more expensive. Since the central authority wallet is used to submit transactions, all have the same *gas* fee. This is an essential topic that can be improved through an associated business model, where users that need a faster blockchain registration can pay more than the other users. This topic will be further analysed in the following section.

Since all the information recorded in the blockchain has an associated cost and is intended to be immutable, it is important to give users the possibility to simply share a file, without recording it on the blockchain. Users can use the share file feature to send a file to another user and keep it in the same platform they used to share important information, without the expenses and formal registry on the blockchain.

In a scenario where the application no longer exists, the file verification can easily be replicated since the hashing algorithm used is public and the smart contract functions to retrieve the file information and their associated users are also public, being only necessary the file hashes as input.

Due to the decentralized and immutable characteristics of blockchain, the information registered on the smart contract can be used as an integrity verification mechanism in, for example, the unfulfillment of a contract or an error in a technical drawing.

In conclusion, blockchain technology adds value to the construction industry because it offers a mechanism of trust between construction participants. However, a business model is required to support the transactions' costs and a central authority is needed to maintain the application, verify the identity of the institutions and register them within the platform. It is also important to give the users a solution that takes advantage of the strengths of the technology, nevertheless, it should not be the only way to use the platform since it might be helpful to allow them to share a file without being recorded on the blockchain. The blockchain network should be public to keep the information recorded on it accessible to everyone, along with the functions to retrieve the smart contract relevant information, depending only on the file and its public hashing algorithm.

4.3 Proof-of-concept

An analysis of the Ethereum transaction execution duration for different *gas* prices has been performed to evaluate two aspects: i) how long does a transaction take to register a file on the smart contract and ii) what is the transaction cost to register that information.

For this analysis, four different *gas* prices were chosen: 1, 10, 50 and 100 Gwei, and was recorded the time spent to execute the *addFile* smart contract function (see Figure 16). Ten sample files were registered consecutively for each *gas* price to minimize the effect of the *gas* price variation over time. The duration was calculated from the moment the Ethereum node is invoked until the transaction receipt is returned. The *gas* price values were chosen to take into consideration the *gas* used to execute the *addFile* function on the smart contract, in order to keep the transaction cost at an acceptable price. The transaction cost can be calculated according to Equation 1 (Ethereum, 2020a):

$$\text{Transaction cost} = \text{gas price} \times \text{gas used} \quad (1)$$

The total *gas* used to execute the *addFile* function is 203 558 units. Table 6 shows the descriptive statistics for ten executions of the same function for each of the different *gas* prices, including the following statistics: minimum, maximum, median, average and standard deviation (obtained in software R).

Table 6 – Descriptive statistics of the transaction duration for different *gas* prices

Transaction duration (seconds)	Gas price (Gwei)			
	1	10	50	100
Minimum	15.55	15.55	15.55	15.56
Maximum	40.23	45.88	30.76	46.09
Median	23.28	15.67	15.77	15.77
Average	24.83	24.72	20.19	21.78
Standard deviation	10.68	12.77	7.27	12.78

It is expected that the transaction duration decreases with an increase in the *gas* price. However, the obtained values of minimum and maximum range between 15.55 seconds (minimum value for 1, 10 and 50 Gwei) and 46.09 seconds (maximum value for 100 Gwei), meaning that in some transactions paying a higher *gas* price does not correspond to a faster transaction. A similar conclusion was reached by Sousa *et al.* (2020), stating that there is no clear correlation between *gas* price and transaction time.

Besides, the standard deviation, when compared with the average and median, shows that the transaction duration has a high variation for the same *gas* price, which can be explained by other factors that influence the transaction duration, such as the usage of the Ropsten test network, the network instability, the Infura API that is used for the Ethereum node and the *gas* price variation through the time, which according to Pierro & Rocha (2019) is strongly affected by the number of pending transactions and the number of miners.

Figure 46 describes Table 6 graphically, showing that the high variation of transaction durations for the same *gas* price is mainly due to the maximum values. This can be proved by the difference between the average and the median values, in which the average values are higher than the median, meaning that the first is influenced by the higher extreme values. For that reason, the median values are chosen to obtain the regression function of the transaction duration for the different *gas* prices (Figure 47).

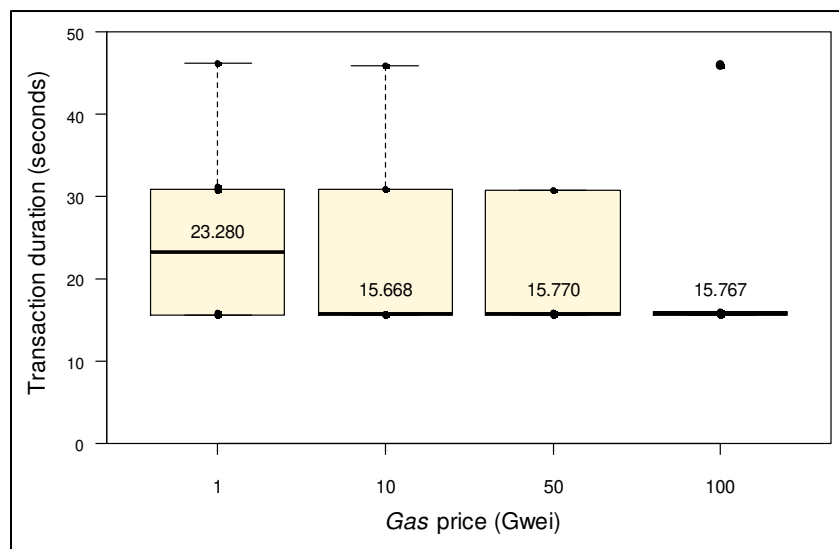


Figure 46 – Boxplot of the transaction duration for different gas prices

The regression function reveals a decreasing trend of the transaction duration for a *gas* price increase. This trend is especially noticed for lower *gas* prices, becoming stable for larger *gas* prices, revealing that for *gas* prices higher than 10 Gwei the transaction time reduction is not proportional to the *gas* price spent.

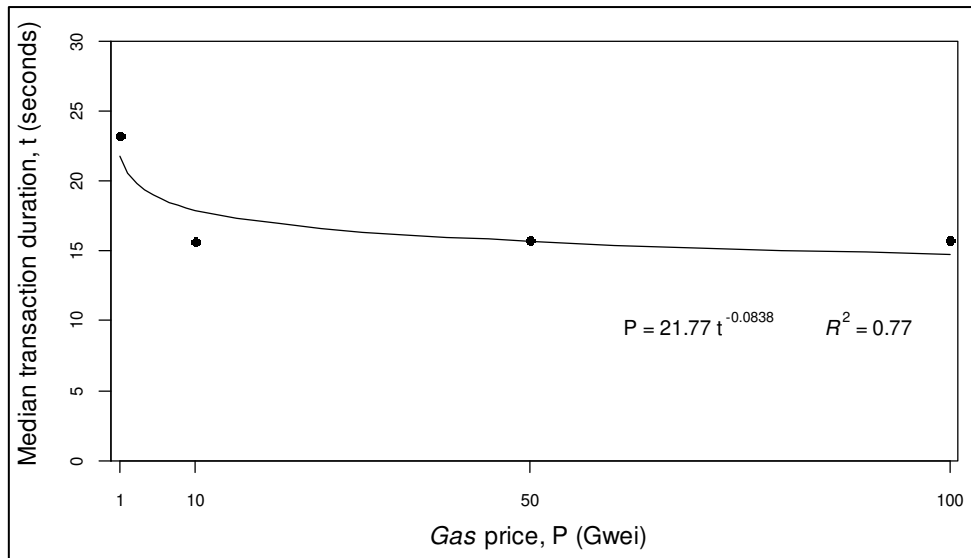


Figure 47 – Regression function of the transaction duration for different gas prices

Table 7 presents the cost analysis of a transaction to execute the *addFile* function in the smart contract. According to Coinbase (2020) was considered that 1Eth is equivalent to 300€ (rate obtained for 26/09/2020).

Table 7 – Transaction cost analysis (rate obtained for 26/09/2020 from Coinbase (2020))

Transaction cost (€)	Gas price (Gwei)			
	1	10	50	100
Ether	0.000203558	0.00203558	0.0101779	0.0203558
Euro	0.061	0.611	3.053	6.107

As the transaction duration does not have a significant reduction for higher *gas* prices, the ideal price to use to execute the *addFile* function transactions is 10 Gwei, which corresponds to, approximately, 0.61 €. Although, since it is a high cost for one transaction, it evidences that a greater effort should be made to optimize the smart contract function, reducing the *gas* consumption needed to execute the function. However, in case the transaction duration is not relevant, the same function can be executed at a lower price (0.061 €).

4.4 Summary

A discussion about the conceptual model has been carried out. It is concluded that blockchain technology adds value to the construction industry because it offers a mechanism of trust between construction participants. However, some concerns need to be addressed: it is necessary to develop a business model to cover the transactions' costs, a central authority is needed to verify the institutions' identity and the information stored in the smart contract should be able to be accessed even if the platform no longer exists.

A POC analysis regarding the transaction duration and cost to execute the smart contract function to register a file was performed. The study showed that the transaction duration is not only affected by the *gas* price, but also by other factors. These factors can be the Ropsten testnet network, the network instability, the *gas* price variation through time and the Infura API that is used as the Ethereum node.

The analysis evidenced that the minimum *gas* price that leads to faster transactions is 10 Gwei, revealing that the smart contract function execution is expensive and showing that an additional effort should be applied to smart contract function optimization.

Chapter 5 - Conclusions and future developments

5.1 Overview

Three major problems in the construction industry, that can be minimized with blockchain technology solutions, were identified in the literature review: lack of accountable information sharing mechanisms, delayed payments and modern contract types. The current dissertation proposes, demonstrates and evaluates a platform to support information sharing between participants of all construction phases based on blockchain technology.

The information sharing platform aims to be the source of information for all of the construction files and simulate physical file sharing with signatures that can later be verified and used as an integrity verification mechanism.

The application allows to share files between the construction participants and record all file interactions in a smart contract deployed on the Ethereum blockchain. The interactions recorded on the blockchain are the upload of new files, upload of a new version of an existing file, signature requests and file signatures. It is also possible to share a file with another user without requesting signatures and, consequently, without recording this information on the blockchain.

A file can be easily verified if was registered within the application, together with its given and pending signatures and their corresponding authors and institutions.

The platform is composed of a central authority that has the responsibility to maintain the application, verify the institution's identity and create them on the platform. Each institution has an admin user that can create the users of the institution and their projects. A project has a set of files that can be into one of the three categories: Technical Drawings, Contracts and BIM model files.

5.2 General conclusions

Blockchain technology can't solve all the technological problems of the construction industry. However, by identifying the key strengths of this technology and the technical problems that the construction industry faces, it is possible to identify some construction issues that can benefit from the application of this technology.

It is concluded that the construction industry can be supported by blockchain technology. The proposed platform allows using this technology to keep an immutable

record of file interaction between construction participants, simulating the behaviour of signatures in a physical document, that can be easily verified even in case the application ceases to exist, being a different solution from the commercial ones available in the market.

A central authority is needed to maintain the platform and to verify and ensure the institutions' identity. This authority is also important to keep blockchain concepts abstracted from the users, with all blockchain interactions, such as Ether acquaintance, being performed by it on behalf of the user.

File integrity verification is a core concept that allows to easily verify if a file was registered on the blockchain, their pending signatures and who signed it. However, it is also important to consider that the file verification should be possible in a scenario where the platform no longer exists. To support this, the smart contract functions to retrieve the file information and their associated users are public, together with the hashing algorithm used for the files. This allows everyone to get the file information by doing a query to the smart contract with the file hash.

Since all blockchain interactions have a cost, it is important to establish a business model around the application. This will be used for the payment of the transaction fees, maintenance of the application and support the central authority in institutions' identity verification. The business model can also be used to differentiate between fast and slow transactions according to the users' needs.

A POC was developed and analysed regarding the transaction duration and cost to execute a smart contract function. The first revealed that the transaction duration is not only affected by the *gas* price, as its reduction is not proportional to an increase of the *gas* spent, but also by other factors. These factors can be the Ropsten test network, the network instability, the *gas* price variation over time and the Infura API that is used as the Ethereum node.

The cost analysis reveals that the execution of the smart contract function to register a file with the ideal *gas* price is expensive, meaning that an additional effort should be made into smart contract function optimization to reduce the *gas* consumed by its execution.

Should be noticed that although the proposed platform intends to support information sharing in the construction industry, there are human factors that influence the

achievement of the desired results. These factors depend on the good usage of the platform and the trust of the users in the system.

5.3 Future developments

In the current dissertation, three construction industry problems were identified. However, the present work focused on a solution for one of the identified industry problems. One future development is the extension of the application to support the delayed payments problem. This will result in a platform that can be used to share information and support payments between construction participants and, therefore, support the new construction contract types.

Another study that can be performed is the evaluation of the adoption of the presented platform in a real construction context and the assessment of the platform's usability.

Since all industries are constantly evolving, another future work suggestion is to evaluate and test new challenges in construction that can benefit from blockchain technology.

References

- Antonopoulos, A., & Wood, G. (2018). *Mastering Ethereum*. O'REILLY MEDIA, INC, USA.
- Asite. (2020). *Asite Common Data Environment*. <https://www.asite.com/project-portfolio-management-ppm/common-data-environment-cde>
- Autodesk. (2020). *Autodesk BIM 360*. <https://www.autodesk.com/bim-360/>
- Baeldung. (2019). *A Comparison Between Spring and Spring Boot*. <https://www.baeldung.com/spring-vs-spring-boot>
- Barbosa, F., Woetzel, J., Mischke, J., João Ribeirinho, M., Sridhar, M., Parsons, M., Bertram, N., & Brown, S. (2017). *Reinventing Construction: A route to higher productivity*.
- Bashir, I. (2017). *Mastering Blockchain*. Packt Publishing Ltd.
- Belle, I. (2017). *The architecture, engineering and construction industry and blockchain technology*. 279–284.
- Bertoni, G., Daemen, J., Peeters, M., & Gilles, V. A. (2009). *Keccak sponge function family main document* (Lecture Notes in Computer Science, pp. 320–337) [Submission to NIST (Round 2)]. <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.419.2140&rep=rep1&type=pdf>
- Bitcoin. (2019a). Block. *Bitcoin Wiki*. <https://en.bitcoin.it/wiki/Block>
- Bitcoin. (2019b). Block hashing algorithm. *Bitcoin Wiki*. https://en.bitcoin.it/wiki/Block_hashing_algorithm

- Bitcoin. (2019c). Block Size Limit Controversy. *Bitcoin Wiki*.
https://en.bitcoin.it/wiki/Block_size_limit_controversy
- Bitcoin. (2019d). Nonce. *Bitcoin Wiki*. <https://en.bitcoin.it/wiki/Nonce>
- Bitcoin. (2019e). Target. *Bitcoin Wiki*. <https://en.bitcoin.it/wiki/Target>
- Bitsusd. (2019). *Bitcoin units converter*. <https://bitsusd.com/bitcoin-units/>
- Black, C., Akintoye, A., & Fitzgerald, E. (2000). An analysis of success factors and benefits of partnering in construction. *International Journal of Project Management*, 18(6), 423–434.
- Bogner, A., Chanson, M., & Meeuw, A. (2016). *A Decentralised Sharing App running a Smart Contract on the Ethereum Blockchain*. 177–178.
<https://doi.org/10.1145/2991561.2998465>
- BootstrapVue. (2020). *BootstrapVue*. <https://bootstrap-vue.org/>
- Buterin, V. (2014). *The Ethereum Wiki*. Ethereum. <https://github.com/ethereum/wiki>
- Buterin, V. (2019). Proof of Stake. *Ethereum Wiki*.
<https://github.com/ethereum/wiki/wiki/Proof-of-Stake-FAQ>
- Coinbase. (2020). *Coinbase*. <https://www.coinbase.com/pt/>
- Das, M., Luo, H., & Cheng, J. C. P. (2020). Securing interim payments in construction projects through a blockchain-based framework. *Automation in Construction*, 118. <https://doi.org/10.1016/j.autcon.2020.103284>
- Ethereum. (2019a). Ether. *Ethereum Homestead*. <http://ethdocs.org/en/latest/ether.html>
- Ethereum. (2019b). *JSON RPC*. <https://github.com/ethereum/wiki/wiki/JSON-RPC>

Ethereum. (2019c). *Programming languages intro*.

<https://github.com/ethereum/wiki/wiki/Programming-languages-intro>

Ethereum. (2020a). *Etherem Development Documentation*.

<https://ethereum.org/en/developers/docs/>

Ethereum. (2020b, September 19). *Etherem Whitepaper*.

<https://ethereum.org/en/whitepaper/>

Etherscan. (2020). *Ropsten Testnet Network*. <https://ropsten.etherscan.io/>

EthHub. (2020). *Intro to Ethereum Wallets*. [https://docs.ethhub.io/using-](https://docs.ethhub.io/using-ethereum/wallets/intro-to-ethereum-wallets/)

[ethereum/wallets/intro-to-ethereum-wallets/](https://docs.ethhub.io/using-ethereum/wallets/intro-to-ethereum-wallets/)

Evan, Y. (2019). *The Progressive JavaScript Framework*. <https://vuejs.org/>

Fielding, R. T. (2000). *Architectural Styles and the Design of Network-based Software*

Architectures [University of California].

https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

Guo, Y., & Liang, C. (2016). Blockchain application and outlook in the banking industry.

Financial Innovation, 2(1), 24. <https://doi.org/10.1186/s40854-016-0034-9>

H2. (2019). *H2 Database Engine*. <https://www.h2database.com/html/main.html>

Heiskanen, A. (2017). The technology of trust: How the Internet of Things and blockchain

could usher in a new era of construction productivity. *Construction Research and*

Innovation, 8(2), 66–70. <https://doi.org/10.1080/20450249.2017.1337349>

Hermes, E. (2016). *Euler Hermes: UK late payments hit two-year high at 2015 year-end*.

[https://www.eulerhermes.com/en_global/media-news/news/press-release-eh-uk-](https://www.eulerhermes.com/en_global/media-news/news/press-release-eh-uk-late-payments-hit-two-year-high-at-2015-year.html)

[late-payments-hit-two-year-high-at-2015-year.html](https://www.eulerhermes.com/en_global/media-news/news/press-release-eh-uk-late-payments-hit-two-year-high-at-2015-year.html)

- Hoepman, J.-H. (2010). *Distributed Double Spending Prevention*. 5964, 152–165.
https://link.springer.com/chapter/10.1007/978-3-642-17773-6_19
- IBM. (2019). *IBM - Advantages of Java*.
https://www.ibm.com/support/knowledgecenter/en/ssw_aix_72/performance/advantages_java.html
- Infura. (2019). *Infura*. <https://infura.io/>
- Investopedia. (2019a). *Hard Fork*. *Investopedia*.
<https://www.investopedia.com/terms/h/hard-fork.asp>
- Investopedia. (2019b). *Soft Fork*. *Investopedia*.
<https://www.investopedia.com/terms/s/soft-fork.asp>
- Java. (2019). *Java*. <https://www.oracle.com/technetwork/java/javase/downloads/jdk11-downloads-5066655.html>
- JSON. (2019). *JSON*. <http://json.org/>
- Lamport, L., Shostak, R., & Pease, M. (1982). *The Byzantine generals problem*. 382–401.
- Lastovetska, A. (2019). *Blockchain Architecture Basics: Components, Structure, Benefits & Creation*. *MLSDev*. <https://mlsdev.com/blog/156-how-to-build-your-own-blockchain-architecture>
- Lin, I.-C., & Liao, T.-C. (2017). *A Survey of Blockchain Security Issues and Challenges*. *International Journal of Network Security*, 19(5), 653–659.
[https://doi.org/10.6633/IJNS.201709.19\(5\).01](https://doi.org/10.6633/IJNS.201709.19(5).01)
- Lisk. (2019). *How does blockchain work*. <https://lisk.io/academy/blockchain-basics/how-does-blockchain-work/what-is-hashing>

- Mathews, M., Robles, D., & Bowe, B. (2017). BIM+Blockchain: A Solution to the Trust Problem in Collaboration? *Gathering*. CITA BIM.
- Mettler, M. (2016). Blockchain technology in healthcare: The revolution starts here. *2016 IEEE 18th International Conference on E-Health Networking, Applications and Services (Healthcom)*, 1–3. <https://doi.org/10.1109/HealthCom.2016.7749510>
- Mozilla. (2019). *What is a webserver?* https://developer.mozilla.org/en-US/docs/Learn/Common_questions/What_is_a_web_server
- Mozilla. (2020). *SPA (Single-page application)*. <https://developer.mozilla.org/en-US/docs/Glossary/SPA>
- Murabito, E. (2019). *Mining difficulty, Hash Power, Nonce Range and the like. An Introduction to Bitcoin Mining*. <https://medium.com/bitcoin-cryptocurrencies-and-blockchain-technology/hash-power-nonce-range-and-mining-difficulty-c4c4e58775f3>
- MyEtherWallet. (2019). *Transactions. My Ether Wallet*. <https://kb.myetherwallet.com/en/transactions/what-is-nonce/>
- Nakamoto, S. (2008). *Bitcoin: A Peer-to-Peer Electronic Cash System*. <https://bitcoin.org/bitcoin.pdf>
- Nginx. (2020). *Nginx*. <https://nginx.org/>
- Penzes, B. (2018). *Blockchain technology in the construction industry (Digital Transformation for High Productivity)*. Institution of Civil Engineers.
- Peyrott, S. (2016). *The JWT Handbook (0.14.1)*. Auth0 Inc. <https://jwt.io/>
- Pierro, G. A., & Rocha, H. (2019). *The Influence Factors on Ethereum Transaction Fees*. 24–31. <https://doi.org/10.1109/WETSEB.2019.00010>

- Pivotal. (2019). *Spring Boot*. <https://spring.io/projects/spring-boot>
- PostgreSQL. (2019). *PostgreSQL*. <https://www.postgresql.org/>
- Remix. (2019). *Remix IDE*. <https://remix.ethereum.org/>
- Sacks, R., Eastman, C., Lee, G., & Teicholz, P. (2018). *BIM Handbook: A guide to Building Information Modeling For Owners, Designers, Engineers, Contractors, and Facility Managers* (3rd ed.). Wiley.
- Sikorski, J. J., Houghton, J., & Kraft, M. (2017). Blockchain technology in the chemical industry: Machine-to-machine electricity market. *Applied Energy*, 195, 234–246. <https://doi.org/10.1016/j.apenergy.2017.03.039>
- Singh, S., & Ashuri, B. (2019). *Leveraging Blockchain Technology in AEC Industry during Design Development Phase*. ASCE International Conference on Computing in Civil Engineering, Atlanta, Georgia. <https://doi.org/10.1061/9780784482421.050>
- Sousa, J. E. de A., Oliveira, V., Valadares, J., Gonçalves, G. D., Villela, S. M., Bernardino, H. S., & Vieira, A. B. (2020). An analysis of the fees and pending time correlation in Ethereum. *International Journal of Network Management*. <https://doi.org/10.1002/nem.2113>
- Swan, M. (2015). *Blockchain: Blueprint for a New Economy*. O'Reilly Media, Inc.
- Szabo, N. (1997). Smart Contracts: Formalizing and Securing Relationships on Public Networks. *First Monday*, 2(9). <https://firstmonday.org/ojs/index.php/fm/article/download/548/469>
- Tapscott, D., & Tapscott, A. (2016). *Blockchain Revolution: How the Technology Behind Bitcoin Is Changing Money, Business, and the World*. Penguin.

- Trimble. (2020). *Trimble Connect*. <https://connect.trimble.com/>
- Turk, Ž., & Klinc, R. (2017). *Potentials of Blockchain Technology for Construction Management—ScienceDirect*.
<https://www.sciencedirect.com/science/article/pii/S187770581733179X>
- Wang, J., Wu, P., Wang, X., & Shou, W. (2017). The outlook of blockchain technology for construction engineering management. *Front. Eng. Manag.*, 67–75.
- Web3j. (2019). *Web3j*. <https://docs.web3j.io/>
- Webpack. (2020). *Webpack*. <https://webpack.js.org/>
- Wong, P. S. P., & Cheung, S. O. (2005). Structural Equation Model of Trust and Partnering Success. *Journal of Management in Engineering*, 21(2), 70–80.
- Wood, D. G. (2017). *Ethereum: A Secure Decentralised Generalised Transaction Ledger*.
<https://pdfs.semanticscholar.org/f65e/e3a9f171da68b57039a5d5f2f1ad70798488.pdf>
- Yang, R., Wakefield, R., Lyu, S., Jayasuriya, S., Han, F., Yi, X., Yang, X., Amarasinghea, G., & Chen, S. (2020). Public and private blockchain in construction business process and information integration. *Automation in Construction*, 118.
<https://doi.org/10.1016/j.autcon.2020.103276>