UNIVERSITY OF SOUTHERN DENMARK

MASTER'S THESIS

Distributed Tamper Proof Task Manager

Author:
Bc. Štefan TÖLTÉSI

Supervisor:
Torben WORM, PhD

A thesis submitted in fulfillment of the requirements for the degree of Master of Science

in the

The Maersk Mc-Kinney Moller Institute
Faculty of Engineering
SDU Software Engineering

June 2, 2019

Declaration of Authorship

“Blockchain is like the new big data or AI - too many people are using it as a buzzword and not focused solving a real problem. We like to call them Blockchain tourists!”

Brad Garlinghouse

UNIVERSITY OF SOUTHERN DENMARK

Abstract

Faculty of Engineering
The Maersk Mc-Kinney Moller Institute

Master of Science

Distributed Tamper Proof Task Manager

by Bc. Štefan TÖLTÉSI

Decentralisation and distribution of various systems is currently a popular trend. An example of this is Bitcoin [1] and other cryptocurrencies. A blockchain technology is on the rise and the fact that we are able to reach a consensus among different parties that do not trust each other may revolutionise how we do business in the future. This advancement has led to many discussions as to how its application can be beneficial to society. [2]

There are a variety of task management tools on the market, but what if we want to make tasks enforceable using smart contracts? None of the current tools provide us with a possibility of using smart contracts and in addition, users or companies have to trust a central third party authority that vouches that the data has not been tampered with. Smart contracts running in blockchain help with removing third party dependency from the system. [3] That is why blockchain technology seems promising when tasks are assigned among non-trusting users that do not want to rely on a specific authority.

The goals of this thesis are following:

- to explore if the task management domain is suitable for utilising blockchain
- to present possible ways of how the blockchain technology can be used in a task management system
- to implement a task management tool that will benefit from properties that blockchain offers

The first two goals are discussed in this thesis and several ways of how to create such a system are presented. One of the ways is chosen and implemented into a basic system for task management. The implementation itself is fulfilled by utilising Hyperledger Fabric blockchain framework.

Keywords: task manager, blockchain, smart contract, Hyperledger Fabric

Acknowledgements

I would first like to thank my thesis supervisor Torben Worm, PhD. Prof. Worm was always helpful whenever I ran into a trouble spot or had a question about my research or writing. He consistently allowed this paper to be my own work, but steered me in the right the direction whenever he thought I needed it.

I would also like to thank Roslyn Vassiliou for helping me with the grammatical side of this thesis.

Finally, I must express my very profound gratitude to my parents and my sister for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them. Thank you.

Contents

List of Figures

List of Abbreviations

API	**A**pplication **P**rograming **I**nterface
EVM	**E**thereum **V**irtual **M**achine
HTTP	**H**yper**t**ext **T**ransfer **P**rotocol
IOT	**I**nternet **o**f **T**hings
IPFS	**I**nter**p**lanetary **F**ile **S**ystem
JSON	**J**avaScript **O**bject **N**otation
PBFT	**P**ractical **B**yzantine **F**ault **T**olerance
REST	**Re**presentational **S**tate **T**ransfer
TTP	**T**rusted **T**hird **P**arty

Dedicated to both of my grandmothers.

Chapter 1

Introduction

In the last few years blockchain has become a buzzword in many industries. The technology is a combination of ideas and concepts from decades of research and enables the creation of a decentralised and distributed ledger that is able to replicate its state among the application's users [4].

Due to its growing popularity, it is important to emphasise that blockchain is not a universal tool that decentralises and distributes every system. The domain itself must satisfy specific properties in order to make a blockchain implementation viable, otherwise the distributed ledger could be replaced by a standard database system [5].

The purpose of the thesis is to address whether blockchain concepts are applicable in a task management context. The market contains a variety of task management software, however the author has not been able to find a solution based on blockchain technology. The thesis will describe the design of a task management system using blockchain technology, documenting the implementation and the specific technologies used.

Structure of this thesis

Chapter 2 explores the theoretical background of the important concepts used throughout the thesis; Task Management, Cryptography and Blockchain.

Following this discussion, Chapter 3 involves a critical analysis of task management. The discussion identifies the current software used for task management, and considers the suitability of its use as a domain for blockchain technology. Additionally, the chapter proposes a solution which could serve as a task management system using permissionless blockchain technology, and addresses the possibilities of saving files in blockchain systems.

Chapter 4 discusses the design of a task management system. To design such a system, requirement specification, together with use case modelling was performed. Then the logical data model was created to specify entities that will be saved into the blockchain. After the specification of entities the complete life cycle of a task is presented. To demonstrate identified task life cycle, a state diagram was created. Finally, this chapter presents the architecture of the task management system.

Chapter 5 describes implemented parts of the final system and highlights used development tools and frameworks used during the implementation. The implemented parts are assembled in one project folder.

The conclusion summarises the findings of the thesis and proposes to whom the implemented solution is useful.

Chapter 2

Theoretical background

In this chapter the most important concepts regarding task management and blockchain are explained. Since blockchain is quite a new technology and resonates in many different industries, it is important to introduce its most crucial parts and their properties. However, before we dive into the blockchain itself we need to understand the domain of task management.

2.1 Task management

Techopedia defines a task management as,

> "task management is an activity in which an individual or team leader tracks a task throughout its life cycle and makes decisions based on the progress."[6]

Task management is a process of managing a task through its life cycle. [7] In organisations, task management is tightly linked with project management, which helps to identify individual tasks in a project that are necessary for completing final objectives. The tasks identified should be related to the organisation's ability to complete a project in relation to cost, time and quality. Therefore, every task should have a set deadline for completion.

The process of task management is essential for every company with multiple employees as it assists with efficiency and co-ordination. By setting up networks of mutually connected tasks, it provides the entity with the ability to plan their operations. As stated in [7],

> "core of a task management is to enable actions of individuals in organisations as well as joint organisational actions. These actions are controlled by knowledge of which the individuals or the organisation dispose. They are driven by goals, for the achievement of which several different ways are usually possible."

Tasks defined in an organisation can be inserted into a task management system, where it is easier to track every detail of the task. These details provide necessary information required for proper planning of the organisation's actions. They also help to identify what tasks a team is working on, to determine the time a task is taking, and to determine the team's efficiency. Most tools allow users to visually manage a task and to see the history of completed, pending, overdue and ongoing tasks. The reports generated by the tools may contain details such as the start date, deadline, overdue date, task budget, main tasks, sub-tasks and time allocation. [6]

2.2 Cryptography

Blockchain heavily relies on fundamentals of cryptography. Cryptography also helps systems to reach properties such as immutability, privacy or non-repudiation. That is the reason why the most important concepts used in further chapters are described here. The basic definition of cryptography by [8] is,

> "Any of various mathematical techniques for encrypting and decrypting data in order to keep it private when transmitted or stored electronically."

These mathematical techniques are important for the whole computer security and are vital for blockchain to serve its purpose. This ensures that the data stored will not be tampered with and that the users can verify the validity of the data stored in a system.

Cryptography has basic algorithms providing a specific functionality distributed into categories called cryptographic primitives. Some of these are crucial in the blockchain system and create a foundation of the blockchain technology. The following section describes the most relevant cryptographic primitives that are used later in the thesis.

2.2.1 Hash function

As stated in [9],

> "A hash function is a computationally efficient function mapping binary strings of arbitrary length to binary strings of some fixed length, called hash-values."

This function should be efficient, therefore the hash should be created quickly.

There is a special class of hash functions called cryptographic hash functions with specific properties. The functions from this class in particular are being used in blockchain. Properties that are required include:

- Pre-Image Resistance
- Second Pre-Image Resistance
- Collision Resistance

Pre-Image resistance ensures that it is computationally infeasible to find an input string for a given output hash value. In other words, it is hard to infer an x for given y when hash function h satisfies $h(x) = y$.

Second Pre-Image Resistance guarantees that it is computationally infeasible to find a second input string that has a same hash value as input string given. So it is hard to find x_2 where $h(x_1) = h(x_2)$ and $x_1 \neq x_2$ when given x_1.

Collision resistance says that it is computationally infeasible to find two arbitrary input strings with a same hash value. That means it is hard to find x_1, x_2 where $x_1 \neq x_2$ and $h(x_1) = h(x_2)$. Since hash function is mapping inputs from an infinite space to a finite set of outputs, it is natural that collisions will occur, however finding such collisions should not be trivial. If this property is fulfilled then the function is also second pre-image resistant. [9]

Cryptographic hash function ensures that an attacker, who wants to find a collision with a given hash, has to invest a substantial amount of computation power.

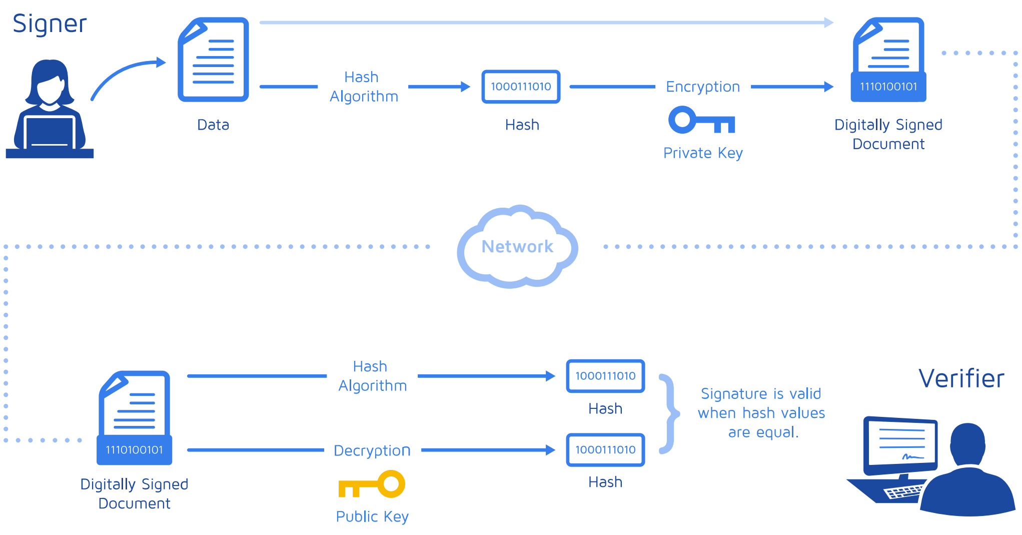

FIGURE 2.1: Process of signing a document and verifying its authenticity [11]

When an attacker finds a collision they can claim that the found fraudulent preimage is genuine.

The cryptographic hash function is later used for creating hashes of files that are enclosed to a task and serve as attachments. Thanks to the function's properties, hash can serve as a fingerprint of a file.

2.2.2 Digital signature

Digital signature is a string which associates a document with some originating entity by using mathematical principles. [9]. Digital signature is (just like a standard hand-written signature) trying to express authenticity of a signed document. As said in [10], digital signatures have the same legal implications as traditional signatures.

Digital signatures rely on asymmetric cryptography. This means two different keys are needed to authenticate a document or message. These keys are also referred to as a public and private key.

Figure 2.1 shows how digital signatures are created and verified. Firstly, the data that is being signed is hashed by a hash function into hash string of a fixed length. Then the hash is encrypted using the signer's private key. This encrypted hash is actually the signature of a document, therefore it is enclosed to the data and sent to the recipient over an unsecured channel.

In order to verify the data received, the recipient creates a hash of the signed data and encrypts the hash with the sender's public key. If the result matches with the signature enclosed, the recipient has successfully verified data integrity. Additionally, this verifies that the sender was indeed an entity with an associated public key.

In task management domain, digital signatures provide a method of securing deliverables against any kind of modification from malicious entities present in the system.

2.2.3 Digital certificate

Digital certificate is a file that helps to establish an identity of a user in a system. It contains user credentials coupled with a public key. This file is verified and digitally signed by a certification authority.

The most used standard for certificates is known as X.509. The most important fields in a certificate are:

- Serial number
- Issuer
- Valid from
- Valid to
- Public key
- Policies

The certificate allows the user's public keys to be securely used in cryptographic functions, therefore a secure channel for communication can be established.

In the scope of this thesis, digital certificates are used for verifying an identity of a user in a system. Thanks to the certificate, the system is able to determine who is performing an action. If all the actions are being recorded it is then trivial to track the user's activity in the system.

2.3 Blockchain

Blockchain by [12] is a linked list using hash pointers. In contrast to a regular linked list where each block has data and a pointer to the previous block in the list, a blockchain replaces the previous block pointer with a hash pointer as can be seen in figure 2.2. These lists are distributed in a network of peers that are actively trying to reach consensus.

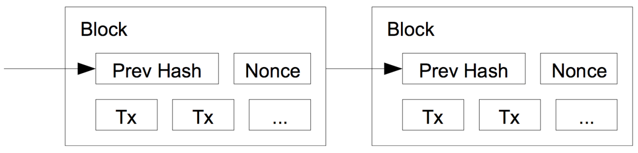

FIGURE 2.2: Blockchain constructed using hashes [1]

An application of this concept that attracted a wider audience is known as Bitcoin. This cryptocurrency was created by an anonymous group of individuals that called themselves Satoshi Nakamoto. In their paper [1], they present the concept of Bitcoin and reveal how they addressed several issues associated with the creation of distributed ledger holding records of transactions.

The purpose of blockchain is to create a data structure that is difficult to be tampered with. When any of the blocks change (or any data within the block), then every consecutive blocks would need to be recalculated, since the hash would differ and the structure would not be valid. That is the reason why solutions using the blockchain principles are tamper-proof.

Blockchain can be split into two categories. The differences are related to who has access to read or write data into the blockchain.

Permissionless Blockchain allows anybody to join the network. Bitcoin and Ethereum could serve as examples. Data saved in the blockchain is publicly verifiable. Since the data is open to public, privacy could be ensured with cryptography. Zerocash, for example, uses zero-knowledge proof for anonymizing transactions. [13] The number of transactions in a specific time interval is limited because of the algorithms for reaching consensus (such as Proof of Work or Proof of Stake).

Permissioned Blockchain grants specific rights and restrictions to participants in the network. Hyperledger [14] or Corda [15] are examples of permissioned blockchains. All users are known and they perform actions under their identities. For reaching consensus, practical byzantine fault tolerant algorithm could be used (PBFT introduced in [16]). PBFT algorithm has bigger throughput of transactions than the ones used in permissionless blockchains. However, PBFT algorithm is vulnerable to Sybil attacks, that is why identity management is crucial in permissioned blockchain systems.

2.3.1 Blockchain vs. centralised system

When considering how to store data in a system with multiple users, there are several ways of achieving this.

One way is traditional centralised system. As a typical example, a database running on a server could be used. Users eligible to send queries can read or write entries in the database. The centralised system is the traditional way of accomplishing persistence.

The more recent solution is blockchain. It inherently distributes data along its nodes, and nodes are actively trying to reach consensus whenever a new entry is being added. There is no central point of failure and there could even be lack of trust among participants thanks to the cryptography.

However, according to [17], the choice for one of the technologies is not trivial. There are properties of a final system that have to be considered before making this decision, since each of the technologies address them in a different way. The most important properties by [18] are:

Public Verifiability allows anyone to check if the current state of a system is correct. In blockchain, every change of state needs to be confirmed by a set of users (miners in Bitcoin context). However, anybody can verify that the changes were made according to a defined protocol.

In a centralised system users may have different views of the state and they cannot verify that the state transitions were executed correctly. Instead, users have to trust that the central authority provides them with the correct state.

Transparency defines how much data will be accessible to an observer. This property is a requirement for a public verifiability.

Privacy is an opposite of transparency. An inherent tension exists between those properties, therefore it is important to specify how much information needs to be publicly accessible. Privacy is easier to achieve in a centralised system, because transparency is not a precondition for a proper functionality of the system.

Integrity ensures that the data retrieved from a system is correct. If the system allows public verifiability, anybody can verify the integrity of the data. In a centralised system the integrity is only guaranteed when a central authority has not been compromised.

Redundancy is important when a failure occurs. Blockchain inherently provides replication among users. In a centralised system, redundancy is achieved through backups.

Trust Anchor states who is authorised in a system to give reading and writing access to other entities.

2.3.2 Application domains

Bitcoin revealed to the world that the blockchain technology is suitable for an implementation of a decentralised digital currency. The application of blockchain principles in the currency domain has been very successful and a significant amount of blockchain implementations have followed Bitcoin's steps.

Other domains where blockchain is being utilised are for example healthcare, IoT and insurance.

There are, however, many domains where blockchain would be suitable for use. To see if a blockchain would be feasible in a certain domain, it is desirable to use the flowchart in figure 2.3 that was suggested in [18]. As can be seen, there are several conditions that must be fulfilled in order to make a domain viable for blockchain technology.

In general, blockchain allows multi-user applications to have a distributed database that can be trusted and is not maintained by a central authority.

Deeper analysis of using blockchain in task management can be found in the next chapter.

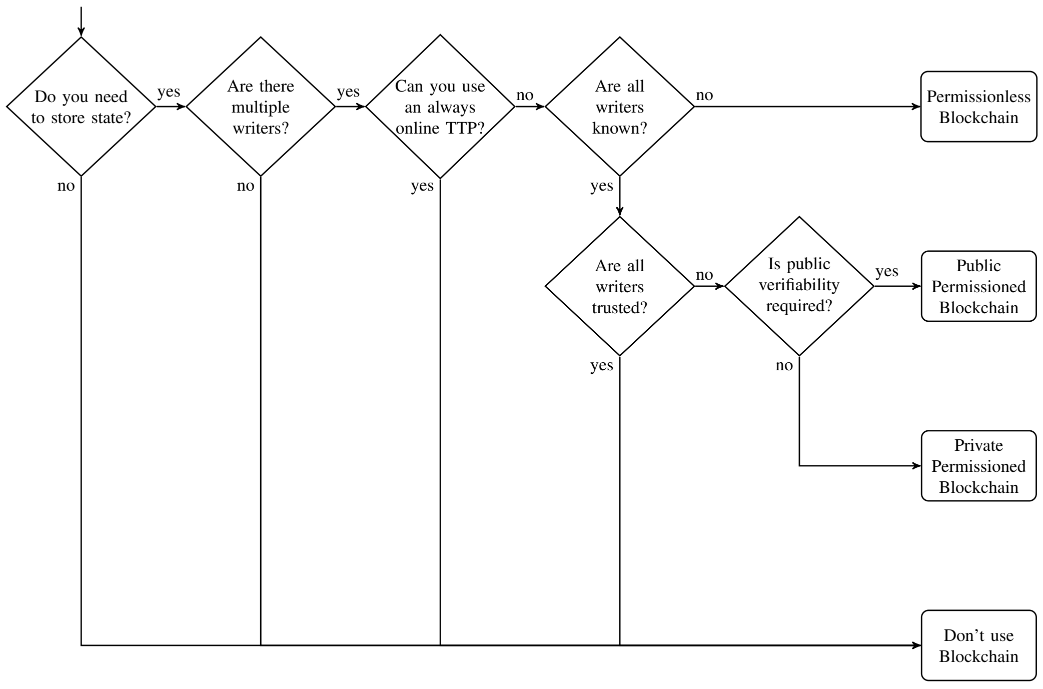
Do you need to store state?
yes
no
Are there multiple writers?
yes
no
Can you use an always online TTP?
no
yes
Are all writers known?
no
yes
Permissionless Blockchain
Are all writers trusted?
no
yes
Is public verifiability required?
yes
no
Public Permissioned Blockchain
Private Permissioned Blockchain
Don't use Blockchain

FIGURE 2.3: Flowchart to determine if a case is suitable for blockchain technology [18]

2.3.3 Smart contract

The definition of smart contract by [19] is:

> "A smart contract is a computer code running on top of a blockchain containing a set of rules under which the parties to that smart contract agree to interact with each other. If and when the pre-defined rules are met, the agreement is automatically enforced. The smart contract code facilitates, verifies, and enforces the negotiation or performance of an agreement or transaction. It is the simplest form of decentralised automation."

The smart contract code runs in the blockchain and in its nodes. In decentralised applications, smart contracts contain the crucial business logic that is expressing what conditions have to be fulfilled in order to perform an action. Smart contracts are the core of such distributed applications. In task management domain, smart contract can help with enforcing the completion of tasks on time and with proper definition of tasks, so they contain all the necessary information.

There are multiple platforms that enable developers to use smart contracts. One of the most used is Ethereum. [20] Developers are using Solidity language to write smart contracts. For execution of the smart contract Ethereum has an Ethereum Virtual Machine (EVM). In order to execute code in the EVM, the user has to provide gas (pay a fee for transaction). If EVM runs out of gas, transaction starting the execution is set as invalid. This is a way of solving halting problem in EVM. Gas is also used to prevent deliberate attacks and abuse of the system [21].

Another platform is Hyperledger developed by Linux Foundation that is backed by IBM. [14] There are several frameworks available. These include Hyperledger Burrow, Hyperledger Fabric, Hyperledger Sawtooth and Hyperledger Indy. In Hyperledger Fabric, smart contracts are called chaincode. Users do not have to pay fees for transactions and chaincode has a predefined time for execution. If this given time is exceeded, chaincode times out. Chaincode in Hyperledger is used to read and modify ledgers state. Since the chaincode runs on a network of peers, it provides a distributed code logic for a system.

2.4 Chapter summary

This chapter provided an overview and discussed the theoretical background of task management and its importance in an organisation. The reader is now familiar with the concept of blockchain, its different types and its most important properties when it is being used for a domain.

Furthermore, the reader was introduced to the foundation of blockchain technology, cryptographical primitives, which help systems achieve information security through their properties. The variety of concepts presented in the chapter are discussed further throughout the thesis.

Chapter 3

Analysis

This chapter analyses the current state of task management software by listing already developed task management software, and by specifying the functionalities this software provides.

Afterwards, the usage of blockchain in task management is discussed. For this purpose, the author uses the methodology proposed in [18]. Use cases and possible solutions utilising permissioned and permissionless blockchain are introduced.

Finally, possible solutions of how to store files in blockchain are proposed, as blockchain is not very suitable for storing large files.

3.1 Existing tools

A significant amount of software systems have been implemented to support all the activities that are required in a project life cycle. Popular examples of this include *monday.com* [22] and *Wrike* [23]. There are multiple reasons why the author of this thesis chose these tools as examples. These systems are highly extensive and have multiple functionalities. They are also very popular. Over 18 000 organisations are using *Wrike*, and *monday.com* registered over 70 000 teams. On the software comparison website *Capterra* [24], both tools have mostly positive reviews.

When author inspected other task management tools, they were similar to mentioned systems. The following sections analyse the already mentioned task managing environments.

Monday.com

Monday.com provides an option to pick from a variety of templates or create custom workflow to get started. Users can sync, plan, organise, and track team projects from the high-level overview down to the smallest details. Over 60,000 teams use *monday.com* to focus on what is important, see who is in charge of what, and stay updated in one collaborative tool. [22]

Users can describe a task and add various attributes that seem appropriate for their use case. Options that can be chosen from are:

- Status
- Person
- Number
- Date
- Timeline

- Link to another task
- File
- and many others

Users can customise what specific attributes are attached to a task which makes the tool highly flexible.

Wrike

[23] *Wrike* can serve as another example. *Wrike* supports the creation of tasks that are assigned to users. After the completion of a task, the task is passed to a reviewer who checks if the task does not need to be adjusted or reworked. The attributes of a task present in the system are:

- Title
- Description
- Assignee
- Due date
- Attachments
- Dependencies on other tasks
- Status

These attributes are always present and the tool is not as customizable as *monday.com*. As can be seen, most attributes are similar for these kinds of tools. These attributes are later used for defining a task in chapter 4.

There are multiple other tools that exist in the market supporting task management processes similar to the ones mentioned above. This kind of software is mostly designed for project management to enable coordination of teams and individuals in an organisation. However, most of these tools require organisations to rely on a trusted third party. The author of this thesis has not been able to find any solutions based on a blockchain technology.

3.2 Blockchain in task management

Whether the task management domain is suitable for blockchain is a fundamental question which must be considered. For this purpose, the flowchart from figure 2.3 is used. Otherwise, the tool developed for task management using blockchain would not bring any new properties to the system and could be easily substituted with already developed software.

The first condition that needs to be fulfilled is that the system has to store state. The task management system must undoubtedly do that as tasks are changing their state during their life cycle.

Secondly, there has to be multiple entities that can write into the system. Task management can be used by individuals to achieve their goals. However, in a group context with multiple individuals, task management is a means for collaboration. Not every individual may have a right to create a task, but anybody can be assigned to complete a task. Especially in larger organisations, there are numerous users performing write actions into the system.

Thirdly, users cannot rely on the fact that trusted third parties will always be available. Also, an organisation may simply not want to give the data away to a third party. Organisations that would use blockchain in their task management system would have direct control over their data.

Additionally, the distinction between permissionless or permissioned blockchain has to be made. If, in the final application, all writers are unknown, permissionless blockchain would have to be used. However, if the system knows an identity of every user, then the permissioned blockchain is suitable.

Since all the conditions are fulfilled for specific use cases in the task management domain, blockchain can be used as a storage layer in task management system. However, there are multiple ways of leveraging blockchain technology in such a system. Ways of applying permissionless and permissioned blockchain are described in following sections, since both blockchain approaches have their advantages and disadvantages. The suitable technology in a specific use case depends on what properties are required from the final system.

3.2.1 Permissionless blockchain

When building a task manager on permissionless blockchain (e.g. Ethereum), anybody would be able to create tasks. However, to assign a task, a public key of an assignee would be needed. In task management, it is necessary to see who is currently working on a task, so the group using this solution would need to have a mapping of group member identities and their public keys. This mapping would be distributed among the individuals in the group via a secure channel. This would ensure that group members that are in possession of such mapping know exactly who the task has been assigned to, and if necessary, could hold the assignee accountable for late delivery.

Everything saved in the blockchain would be visible to everybody, therefore the system would be highly transparent. However, if the data saved would be private in nature, they would need to be encrypted. For example, data would be encrypted with symmetric cipher, where the key would be a shared secret of all members in a group (see figure 3.1) . The shared secret would be a result of a Diffie-Hellman key exchange algorithm. This would allow any member of the group to read the private data.

Another way of ensuring privacy and using the blockchain as a tamper-proof element in the system would be storing only hashes of the private data. For storing the actual data, different storage would be used (see figure 3.2). To keep balance between transparency and privacy, data that is not confidential would be stored into the blockchain together with the hash of the private data. The type of storage determines the properties of the final system. Further description can be found in section 3.3.

Integrity of the data would be ensured by using a blockchain. The longer the blockchain is, the higher the probability that the older data entries have not been tampered with. However, the system is vulnerable to the attacks aiming on its consensus algorithm. For example, in proof of work algorithm, if an attacker controls more than half of the nodes present in the blockchain network, the transaction confirmation can be entirely blocked by the attacker [25].

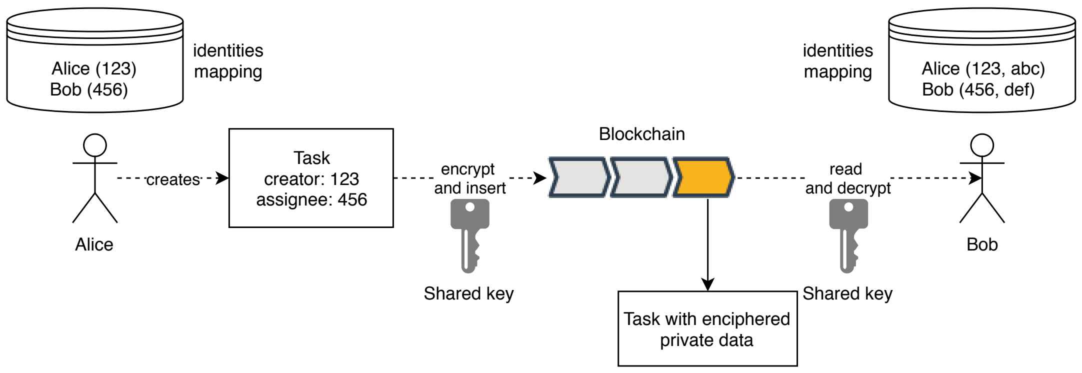

FIGURE 3.1: Process of inserting task with private information into a blockchain

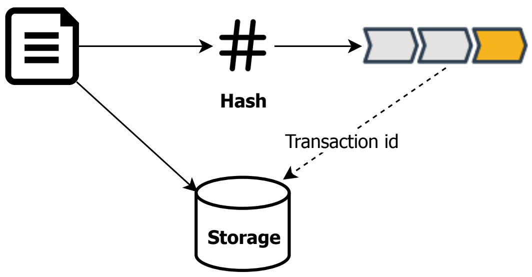

FIGURE 3.2: Hash of the private data is stored in blockchain. Data itself is stored in non-public storage together with id of blockchain transaction.

Redundancy in the system is inherently ensured by blockchain as well. Data in the blockchain is being replicated along the nodes. So losing data on one node does not mean that the data is lost from the system.

Trust anchor for the solution proposed above would be an entity charged with the identity mapping distribution.

One weaknesses of a system built on permissionless blockchain would be the speed of transactions. Compared to practical byzantine fault tolerant algorithm, the most used permissionless blockchain consensus algorithms have lower transaction confirmation throughput. The limit is dependent on the blockchain network and consensus algorithm that is used. [26]

Another limitation is that in order to use a public permissionless blockchain network, users have to pay fees to miners in order to validate transactions and reach consensus among nodes. The Ethereum fee that has to be paid is called gas. In the future the gas price might rise and that would result into significant financial burden on parties using proposed solution.

In general public permissionless blockchain would perform well in projects where high transparency is required. For example, some governmental projects could use task management tool that is built on permissionless blockchain and achieve high transparency by doing so.

Another example could be crowdfunding projects. By keeping the flow of a task public, stakeholders (backers of the project) could see the development process of the project and check if the organisation is spending its resources wisely.

3.2.2 Permissioned blockchain

Private companies typically do not want to give away any information to wider audiences for various reasons. These companies that do not require public verifiability in their task management system and which create digital identities for their employees, can benefit from putting tasks into a private permissioned blockchain. By doing so all the changes made in the system are recorded in a blockchain. Only participants with valid certificates can access data and perform actions in the blockchain by using smart contracts. There are no transaction fees needed in order to run smart contracts in permissioned blockchain.

This thesis proposes a tool that can be used in an organisation for task management. For the implementation of a final system, permissioned blockchain was chosen as storage layer, since private companies have a set hierarchy and want to control access of users to different parts of their systems. Therefore, private permissioned blockchain is the best choice for such a system.

Public verifiability is not present in the proposed solution, since only invited entities have access to the blockchain. Therefore, only these entities can verify the blockchains integrity.

As a blockchain platform Hyperledger Fabric was chosen because of several reasons:

- smart contracts (called chaincode in Hyperledger) can be developed in various general purpose languages (Go, Javascript, Java)
- modular architecture for pluggable consensus and membership services [14]
- no built-in cryptocurrency
- extensive documentation

Other permissioned blockchain platform that is available is R3 Corda [15]. However, this platform is more suitable for financial service industry, as it was specifically developed to operate in financial domain.
The solution is addressing basic task management system that can be used within organisations for project management. However, before starting the design chapter of such a system, possibilities of how to store files in a blockchain are discussed.

3.3 Storing files in blockchain

In task management, there is often a situation where task has an enclosed document or a file that needs to be modified to complete the task. Also, task completion may expect some kind of deliverable attached to the finished task. Therefore, task management tool has to support this kind of functionality.

However, saving files into a blockchain is not a trivial problem. The files can be saved directly into the blockchain, but since blockchain replicates its state among other nodes in the network, files would also be saved in every node present in the network. This would result in enormous blockchain size in the future.

Additionally, when using permissionless blockchain, saving data is very expensive. For example, according to Ethereum yellow paper [27], the fee is 20k gas to store a 256 bit word in the blockchain. For a gas price 50 Gwei this would make transaction fee of 0.165$. [28]

A solution is to store the data elsewhere and to store the hash of the data in the blockchain. That is the same principle that was presented in section 3.2.1 for handling private data in permissionless blockchain. Figure 3.2 shows how a file is saved in a different storage together with the id of transaction saved in the blockchain. What type of storage is used influences the properties of a final system.

The storage can be a relational (or objective) database. It is fast to query for data and it is a well known solution for storing large amount of data. However, by using this kind of database, decentralisation would be lost and the central point of failure would be brought back into the system.

To keep the system distributed, a distributed file system could be used. A file is distributed among nodes, and users can access them if they know its URL. However, the file is not saved in every node of the system. Only a subset of all nodes are keeping the file and provide it to other users. An example of distributed file system is Inter-Planetary File System (IPFS) [29].

3.4 Chapter summary

In this chapter, existing solutions addressing task management and project management have been presented. Author of the thesis was not able to find task management software based on a blockchain. Therefore, suitability of the blockchain technology for task management had to be discussed.

The above analysis identified that blockchain technology would be useful in specific cases of task management; where trust among users and availability of a third party is an issue. Otherwise, the system can be implemented by using traditional technology (e.g. relational database).

One of the proposed solutions would be to use permissionless blockchain technology. Alternative solution using permissioned blockchain, suitable for private organisation, is used for design and implementation of the final system.

Finally, a way of storing files in the system based on blockchain has been proposed, since task management domain often requires files attached to the identified tasks.

Chapter 4

Design

This chapter describes a design of task management tool based on permissioned blockchain. The system is designed for a large organization with several departments that can have an external developers and suppliers. Every entity has a certificate to determine the entitie's identity. The design is for minimal viable product that could be used in task management in an organization.

4.1 Requirement specification

First specification of functional and non-functional requirements is stated. These are basic requirements for a task management tool that can be used in organization for project management. These requirements come from the analysis of the already created tools discussed in chapter 3. Requirements also reflect that the system is made for private organizations that need a decentralized solution by utilizing private blockchain technology.

4.1.1 Functional requirements

- Users are able to create tasks.
- At any point of time it is clear in what state a task is currently in.
- Chains of tasks can be created in the system.
- Every modification or change of state is recorded in blockchain.
- Tasks contain all the necessary information for their completion.
- System provides an overview of all the tasks.
- System provides a decentralized shared state for all users.
- Corruption of data is detectable.
- System supports revision of tasks.

4.1.2 Non-functional requirements

- System is modular and enables an organisation to change parts of the system.
- System is scalable, since number of employees in an organisation (users of the system) fluctuates.
- System is fault tolerant to maintain availability of the system.
- Consensus has to be reached in the matter of seconds.

4.2 Use case modelling

This section defines what entities are present in the system and displays the interactions of users and the final system. A use case diagram has been created for this purpose. Use case modelling helps with capturing what the system is supposed to do and stating what functionalities should be present.

4.2.1 Actors

First it is important to specify what entities will be using the final system. These entities are called actors in use case modelling. [30] The specification helps to clarify the roles that different users can have. Different actors may have different functionalities available to them. Therefore, system has to differentiate users according to their roles. For task management it is important to identify a user that is completing the task and a user that assigned the task. A description of identified actors that will interact with the system follows.

Participant

Anybody who will be using the system for task management. The participant represents a common user. The participant needs to posses verified digital certificate to communicate with the system. The digital certificate is provided by an organisation governing the system. By looking at the certificate the system determines user's identity.

Admin

User that has access to all the data stored in the blockchain. Defines and maintains the network of nodes collaborating in the blockchain. Has to be a verified user by the organisation, since admin performs crucial activities to ensure that the system is running. Admin can perform any of the actions available to task owners and participants.

Task owner

A participant that created a task is a task owner. However, task ownership can be transferred to a different participant. The task owner modifies and reviews the tasks.

Assignee

A participant that was assigned to complete the task. Can send a task for revision or choose a task for completion.

4.2.2 Use case diagram

Figure 4.1 shows use cases of the proposed system. A description of presented use cases follows.

UC1 Actor admin can deploy chaincode into a network.

UC2 Once the chaincode expressing the main business logic is deployed, it can be updated to fix eventual bugs or to change the business logic itself.

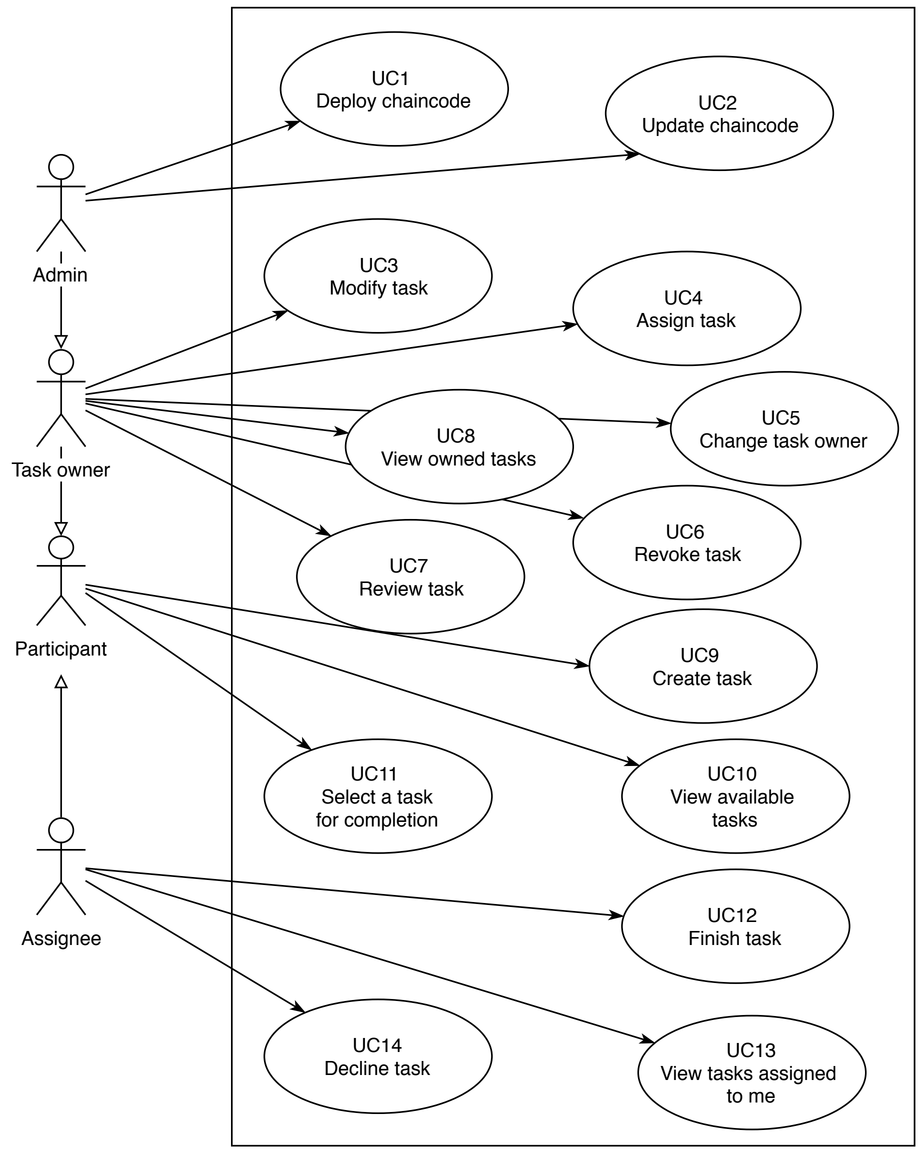

FIGURE 4.1: Use case diagram of a simple task manager

UC3 Task owner is able to modify details of already created tasks that have not been assigned to a participant yet.

UC4 Task owner can assign a participant to the task.

UC5 Task owner can transfer an ownership of a task to another participant.

UC6 Task owner may revoke an assignee from a task.

UC7 When task has been completed, a task owner needs to review if an assignee had completed everything that was stated in the task description and delivered all the deliverables.

UC8 Task owner can see all their currently owned tasks.

UC9 Participant can create a task and automatically becomes an owner of that task.

UC10 Participant can see all the unassigned tasks.

UC11 Participant can select a task from a list of unassigned tasks and become an assignee of that task.

UC12 Assignee can finish the task when it has been completed.

UC13 Assignee can see what tasks they have been assigned to.

UC14 Assignee can decline a task that they have been assigned to. Task returns to the list of tasks without assignee.

4.3 Task as a model

After analysing common attributes of tasks in various task management tools in section 3.1, this part of the thesis shows the most important attributes of the tasks.

A logical data model of the data saved in blockchain can be seen in figure 4.2. Task contains all the necessary attributes to support basic task management activities. List of attributes and their description follows.

Id	identifies a task. The id is used for retrieval of the data saved in a blockchain.
Title	is a short description of a task. Helps users to quickly identify a task in lists of tasks.
Description	provides a detailed information about the task. Defines deliverables and specifies actions that needs to be done before task completion.
State	holds an information about the current state of the task. Further description of states can be found in the next section.
Task owner	is an identifier of participant that owns the task. A task has an owner at all times. When a task is created, the creator becomes an owner of that task.
Assignee	is an identifier of the participant that has been assigned to complete the task. Task owner either assigns a participant to a task or a participant selects a task from a list of available tasks by entering assignee's id.

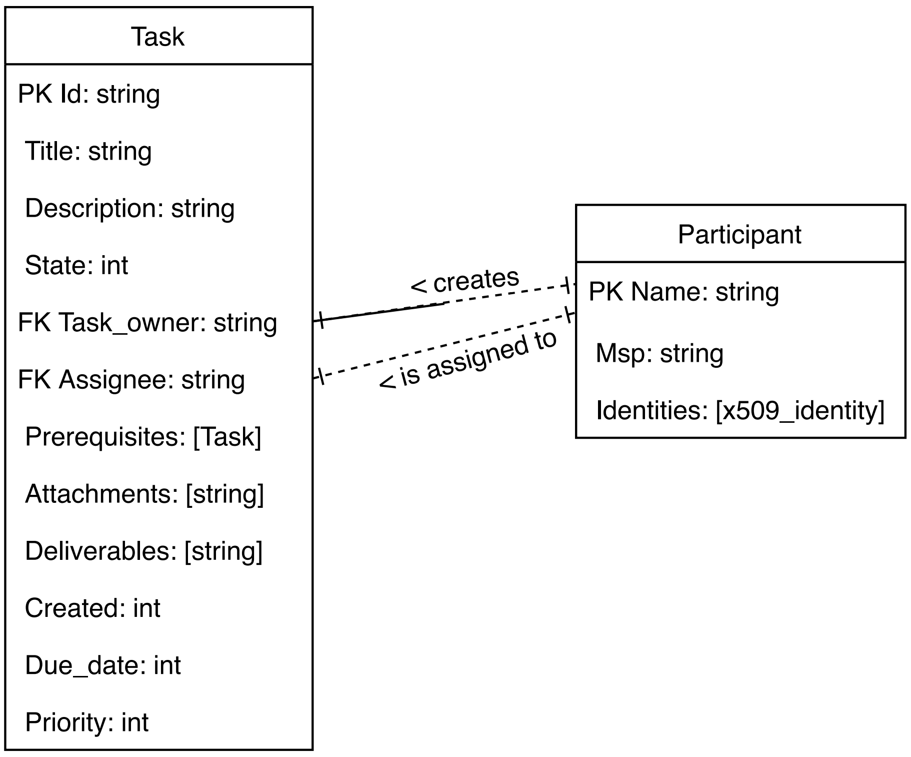

FIGURE 4.2: Logical data model for basic task management system

Prerequisites	hold a list of identifiers that identify tasks that serve as prerequisites for the created task.
Attachments	contain hashes of files that serve as a supplement to task definition. As discussed in section 3.3, only the hash of a file is stored in a blockchain and the file itself is stored in a different storage. Task may have multiple attachments. By having this property, the system is ready to support file verification and it gives a certain degree of confidence that the task has not been modified. It is important that the hash is created by a cryptographical hash function.
Deliverables	similarly to attachments, they contain hashes of files. These files are delivered when the task is completed.
Created	is a timestamp of when the task was created.
Due date	specifies a date that is considered a deadline for the completion of the task.
Priority	specifies how important the task is. Used for planning and prioritisation.

Another entity saved in blockchain is a participant. This entity represents the user and their identities in the system. Attributes saved for the participant are in the following list.

Name	is a username that identifies a user in a system.

Msp stands for membership service provider. It is an identifier that specifies who is the certificate authority for issuing certificates.

Identities is a list of x509 certificate identities. Each of these identities contain an information if the certificate is still valid and a fingerprint of a certificate. The certificate determines a role of the participant in the system.

4.4 State diagram of task

As tasks change their state throughout their life cycle, the individual states have to be identified and the transitions from one state to another have to be presented. To demonstrate a life cycle of a task, the state diagram that can be seen in the figure 4.3 was created.

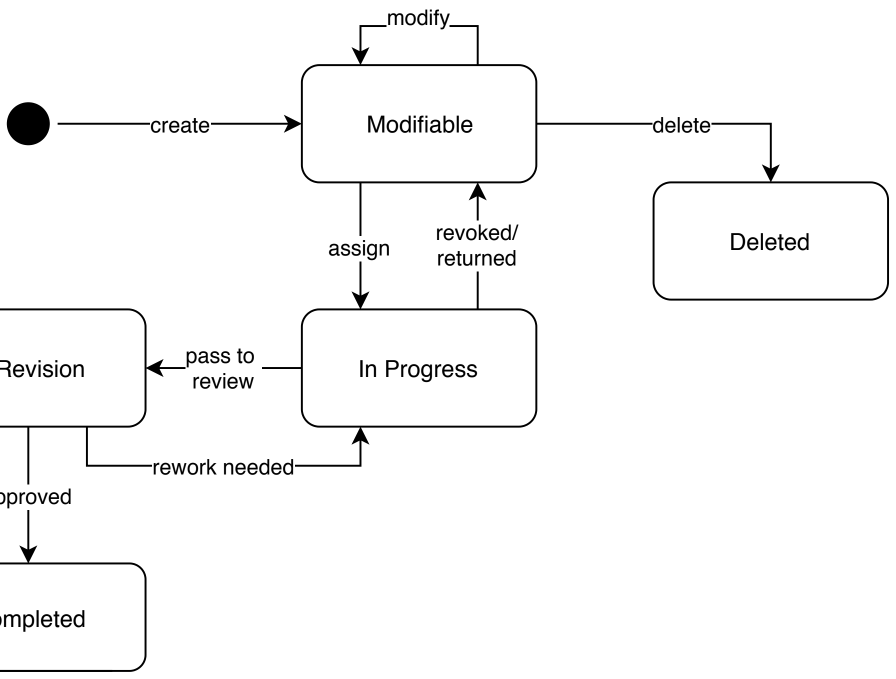

FIGURE 4.3: State diagram showing life cycle of a task

The whole cycle starts when a participant creates a task by providing all the required information. This transitions the task into a *Modifiable* state.

Modifiable state is when the task has been created, but there are no participants assigned to it. The task owner can still modify the details of the task.

When in *Modifiable* state, a task can be deleted from the system. This does not mean that the task is completely removed from a blockchain. All the transactions that modified the task are still recorded in the blockchain. However, the task itself is no longer present in the ledger.

When a participant is assigned to a task, the task transitions into *In Progress* state. When the assignee is not able to complete a task it can be returned to a *Modifiable* state by the assignee or the task owner by revoking the task from the assigned participant.

When an assignee finishes the task, it is passed to the task owner for revision. The task owner revises if everything stated in the task description has been completed. If there are problems, the task can be sent back to an assignee for rework. However, if everything has been completed and delivered successfully, the task can be moved into a *Completed* state.

When a task is retrieved from the system, a user can quickly determine in what state the task is in by simply looking into its state attribute.

4.5 Architecture of the system

The whole system is build upon a Hyperledeger Fabric platform. The figure 4.4 represents an overview of the architecture of the final system. As can be seen, the whole system has multiple layers that interact with each other.

The lowest layer where data is kept consists of multiple nodes. These nodes, ensuring that a consensus is reached, are owned by the organisation's departments or external contractors. Nodes are communicating over channels. Each of these channels has their own ledger that is shared among nodes. This shared ledger is identical for every node in the channel. It is also a place where all the transactions are saved. Every node also has its own certificate that is used for secure communication between nodes and to determine role of a node.

Chaincode installed in nodes contains the business logic for task management and interacts with the ledger whenever data is being stored or retrieved. Chaincode can recognise the caller by looking at its certificate. Therefore, the chaincode determines if the function called will be performed or not.

Part of a system that is performing chaincode calls is *Applications/Consumers* layer. In the implementation, this application is a server application that provides a REST API to the front-end part of the system.

The layer that the final user interacts with can be a web application, a terminal window or a mobile application. Participant using the web application is fully abstracted from the fact that the system is running on a blockchain. This front-end part is not included in the implemented solution.

Participants can run the application that invokes chaincode methods on their machines. The application is associated with a certificate that determines caller's identity and their permissions. Alternatively, the application could run on an organisation's server and participants would have to sign in with their credentials in order to use the system. This way, the server application would find out user's identity, and it would be able to select an appropriate certificate that is associated with the user.

One of the advantages of the proposed architecture is that it supports full modularity of each of the layers and it gives a full control of the nodes to the organisation and its departments. It also enables the organisation to define its specific rules for reaching consensus among nodes. Another advantage is that the transactions are processed in a matter of seconds.

The disadvantage of the architecture is the fact that the organisation has to start managing certificates in order to use the system. The whole system is also quite complex and requires skilled staff to maintain the network of nodes.

When the proposed architecture is compared to a centralised system, it is obvious that there is no single point of failure, since the chaincode is distributed among nodes along with the ledger and its data.

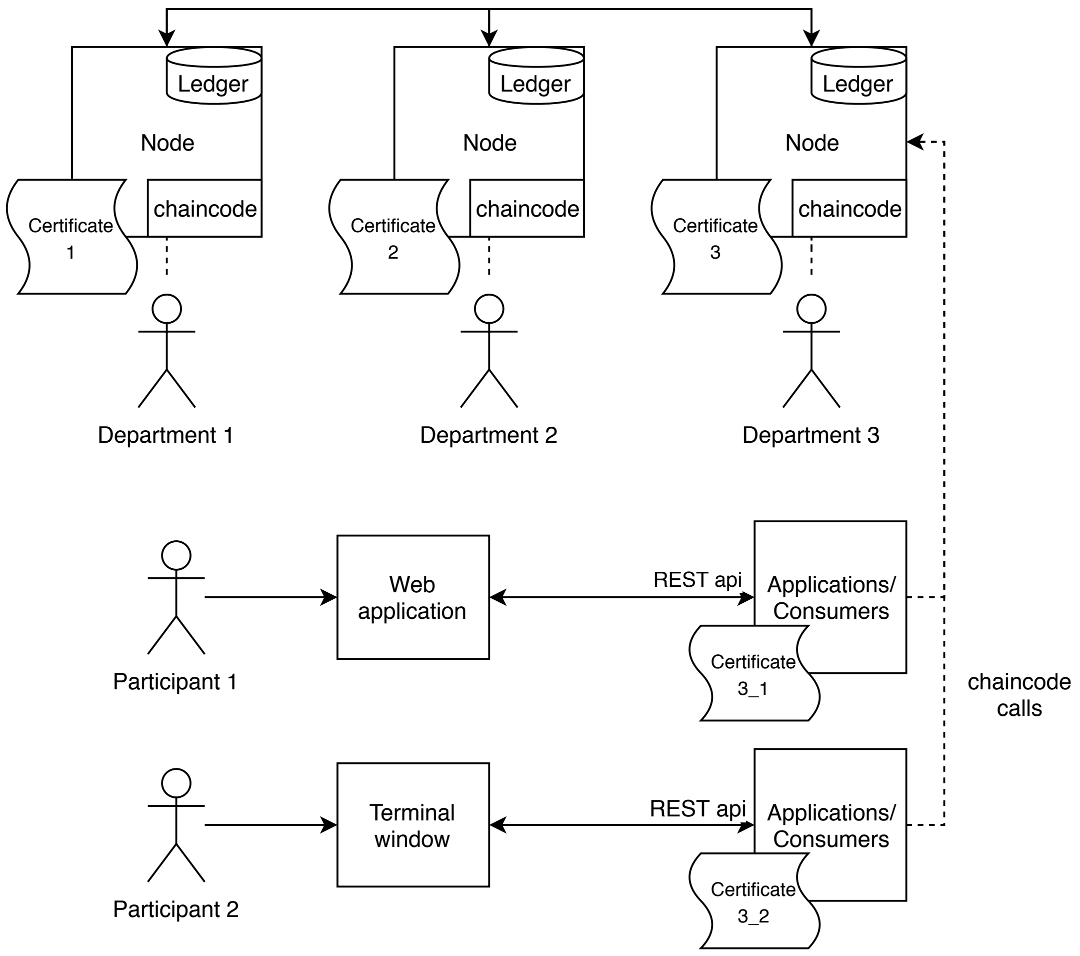

FIGURE 4.4: Diagram showing the architecture of the system

4.6 Chapter summary

This chapter specified basic system requirements for task management system build on permissioned blockchain, followed by the use case modelling that identified the actors present in the system and the activities they can perform. As a next step, the most important attributes of a task were identified. This resulted in creation of logical data model. To present the task life cycle in the system, a state diagram has been created. Finally, an architecture has been proposed that supports the distribution of task management system by using permissioned blockchain. Architecture has multiple layers to support modularity and achieve separation of responsibilities. The final users do not have to be aware that the implemented solution is having a blockchain technology as a storage layer.

Chapter 5

Implementation

This chapter discusses the implemented parts of the task management system. The core of the whole system is the developed chaincode that has been tested by unit tests. Another part of the system is a Node.js application that has been developed to invoke the chaincode under an identity of a supplied certificate. It provides a REST API to the front-end part of the system. However, prior to the implementation, it is important to understand the development tools and frameworks used for the implementation.

5.1 Development tools

Multiple development tools have been used for implementation as they simplify the development process of the system. The platforms mentioned below are related to development of permissionless blockchain.

5.1.1 Hyperledger Fabric

The most important platform is Hyperledger Fabric, as it is a permissioned distributed ledger technology platform designed for use in enterprise context. The whole platform is open source, modular and enables developers to create a permissioned network of known participants. Algorithms for identity, consensus and encryption can be plugged into a network, allowing the platform to be easily customisable and allowing organisations to tailor the platform to their specific needs.

The created blockchain is a distributed system consisting of many nodes that communicate with each other. The blockchain runs programs called chaincode, holds state and ledger data, and verifies transactions. The chaincode is a central element of the system, as it contains business logic and serves as a model layer. Transactions have to be "endorsed", and only endorsed transactions may be committed and have an effect on the state. [31]

The ledger itself is comprised of a world state and a blockchain, as can be seen in figure 5.1. The world state reflects a system after applying all the endorsed transactions that are saved in the blockchain. It holds all the current values present in the system and these values are saved as a key-value pairs.

Nodes running the system are containerized and their images are run via the Docker platform. As the Docker images have the necessary dependencies required for running a node, it is easy for an organization to start a node in a Linux or Windows environment.

However, while starting a node is not complicated, setting up the whole network is still cumbersome in Hyperldeger Fabric. The platform is still evolving and developers have to put in a lot of effort to set up a developer environment that enables

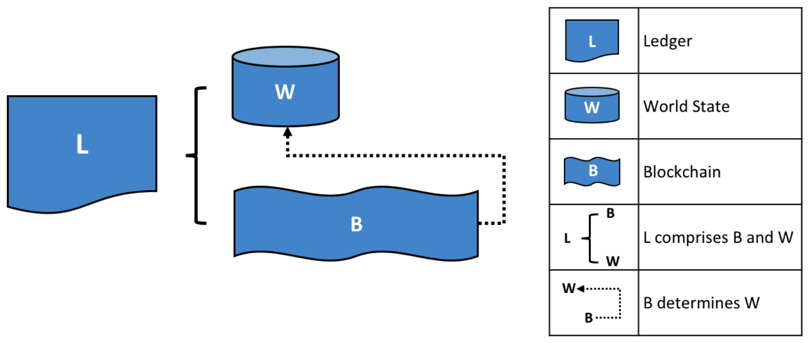

FIGURE 5.1: Ledger in Fabric consists of a world state that is derived from blockchain containing endorsed transactions. [31]

them to easily test and deploy the written code. This is the reason why there are several frameworks making the whole platform more accessible.

5.1.2 Hyperledger Composer

Hyperledger Composer is a set of tools for building blockchain business networks that make it simple and fast for business owners and developers to create smart contracts and blockchain applications. [32]

Composer provides an additional layer over Hyperledger Fabric and enforces developers to model assets, transactions, participants, and events by using Hyperledger Composer Modelling Language.

The first version of the system implemented for this thesis was written in Composer. However, the creators of Composer began to focus on bringing new features directly into the Fabric platform, thus they stopped releasing new features for the Composer itself. For this reason, the initial logic written in Composer Modelling Language system was abandoned. The implementation then focused on using a new set of tools called Convector Suite.

5.1.3 Convector Suite

Convector Suite is an open source suite for enterprise blockchain networks. By the official documentation [33] its main components are:

Convector Smart Contracts is JavaScript-based development framework for enterprise smart contract systems.

Hurley is the easiest way to quickly setup Hyperledger development environment. Instead of learning all the configuration files required, a developer can just use Hurley to setup the environment for smart contract development.

Convector CLI is the fastest and easiest way to build a new Convector Smart Contracts project. CLI is fully integrated with Hurley.

When compared with Composer, Convector does not provide an additional layer over Hyperledger Fabric. Developer uses Convector to work with the native Hyperledger. This way, projects are not limited by compiler that would need to compile a model language into Hyperledger Fabric code as in Composer.

Convector encourages a model/controller architectural pattern and the code written in this thesis is conforming to the pattern as it makes the code easier to modify and promotes re-usability.

The following section discusses the implemented parts and contains relevant code snippets.

5.2 Implemented parts

This section presents the parts of the final system that have been implemented during this thesis. There are two main parts that have been completed. The first is a chaincode which contains the main business logic. The second is a Node.js application that invokes the written chaincode and provides data to a front-end part of the system via REST API.

Both parts are combined in one project called *tasknet-convector*. Further details on the project structure can be found in section 5.4.

The first described part of the system is the chaincode that will run on a distributed blockchain network.

5.2.1 Chaincode

Chaincode is the main part of the implementation and it is essential to the final system. The code itself is written in Typescript, which is a typed superset of JavaScript language. Typescript brings static typing into JavaScript environment.

The code base of the chaincode consists of two different packages. However, prior to installing the chaincode onto the blockchain network, both packages are bundled into one package using *chaincode-manager* tool from Convector. The newly created package is then installed by using Hurley.

The individual packages are:

- Task
- Participant

Task package contains all functions related to task management and handles various task states. The definition of a task data structure is located in *task.model.ts* file. The data structure is designed in accordance with the data model presented in figure 4.2. Validation of all properties is done by using *yup* npm package. When property does not pass, the *yup* validation program automatically throws an error. Figure 5.2 shows a validation of an array of strings.

```
        @Validate(yup.array().of(yup.string()))
        public prerequisites: string[];
```

FIGURE 5.2: Validation for prerequisites using *yup*

Controller is located in *task.controller.ts* file and provides the main functionality for the system and stores and retrieves data structures from the Hyperledger's ledger. As the task model and controller extend Convector model and controller classes, saving and retrieving data is straightforward, as can be seen in figure 5.3.

The implemented methods of controller follow the state diagram flow presented in figure 4.3. There are also additional *get* methods present for retrieving the data

```
        const task = await this.getTask(id);
        await task.save();
```

FIGURE 5.3: Example of how to retrieve and store data in the ledger using Convector

from the ledger. Following list presents all of the implemented methods and their description.

create is a method that creates a task in the system. The method initialises a task object with values that were sent as parameters. Any participant can create a task.

modify is a method for modification of created tasks. The task has to be in *Modifiable* state. System admin or the owner of the task can invoke this method.

assign is used for assigning a participant to the task. The assignee is supplied in parameter as a string identifier of a participant. System admin or the task owner may use this method.

saveDeliverables is a method for saving a hash of deliverables. The actual files will be saved in an alternative storage. System admin or an assignee can save deliverables to the task.

passToReview is used for confirming that the work on the task has been finished. The task is then passed for a review to the task owner. System admin or an assignee can pass task to the review.

approve is used for confirming that the task has been finished and reviewed. System admin or the task owner can approve the task.

revoke is for revoking a task from an assignee. A task owner is able to revoke an assignee from a task or an assignee may reject the task. In either case, the task returns back into *Modifiable* state and a new participant can be assigned to the task.

rework sends the task back to an assignee. System admin or the task owner can send the task back to the assignee for a rework.

transferOwnership is a method for transferring an ownership of a task to another participant. System admin or the current task owner can transfer an ownership of the task.

delete removes a task from the ledgers world state. System admin or the task owner may delete a task.

get retrieves a task with a specific id. The task can only be retrieved by its owner, assignee or a system admin.

getOwned retrieves all tasks that are owned by the participant. The callers identity has to be in the list of associated identities for the passed participant.

getAssignedTo retrieves all tasks the participant is assigned to. The callers identity has to be in the list of associated identities for the passed participant.

getUnassigned retrieves all unassigned tasks in the system.

getAll is used by the system administrator to retrieve all tasks from the system.

The code of a second package called *participant* was reused from a Convector example project that addresses identity patterns in the project [34]. By using the *participant* package users have their identities mapped to the active certificates they are using. If a certificate gets compromised, the identity in the system can be invalidated and a new identity with a new certificate can be registered. By decoupling identities and participants, other packages do not need to be concerned with the changes that are made to the identities. Therefore, *task* package contains an id of a participant and when an identity for the participant changes, there will be no change from *task* package's perspective. The implemented function in figure 5.4 shows how a participant (supplied by the function parameter) is being matched with callers identity. The code on line 7 is the direct comparison of participants active identity and callers identity.

```
1  private async participantIsCaller(participantId: string) {
2      const participant = await Participant.getOne(participantId);
3      if (!participant || !participant.id || !participant.identities) {
4          throw new Error(‘Participant with id: "${participantId}" does
               not exist.‘);
5      }
6      const activeIdentity = participant.identities.filter(identity =>
           identity.status === true)[0];
7      if (activeIdentity.fingerprint === this.sender) {
8        return true;
9      }
10     return false;
11 }
```

FIGURE 5.4: Functions that detects if a callers identity belongs to a participant.

When the caller of the chaincode is not allowed to perform a certain action or they invoke the chaincode with invalid parameters an error is being thrown. An example of throwing an error can be found in figure 5.4 on line 4. In this case the caller has invoked the chaincode with a non-existing participant id.

The created chaincode can be installed onto a Hyperldeger's node network and serve as a distributed database for task management software.

5.2.2 Node.js application

A second part of the implementation is Node.js application that provides a REST API that is used for communication with the chaincode. The base structure of the application was generated by *convector-rest-api* [35] software. The tool was recently released and as a result, bugs were still present. The code generator was not able to correctly generate endpoints, where parameters contained values of enumerations. The author of the thesis contacted the developers and they were able to fix the encountered issue and the current version of the tool should work as originally intended.

The structure of the API and its endpoints are linked with each of the invokable methods present in the chaincode (these methods are listed in section 5.2.1). To mark a method for having its endpoint in the final API, special decorators must be used. These decorators specify if the generated endpoint will expect *GET* or *POST HTTP* request method.

For the API definition, Swagger has been utilised. The author of the thesis modified multiple files to allow the Node.js application to support all of the methods of the chaincode, since the *convector-rest-api* generator does not support generation of multiple *get* methods for one model class defined by the chaincode.

Users with certificates registered in the system may use the created application for chaincode invocations. However, the application requires a user's personal certificate that is provided by the organisation managing the system. This way the application performs invocations of the chaincode with the identity supplied by the certificate.

5.3 Testing

Implemented chaincode also contains unit tests. To create these tests a popular JavaScript framework named *Mocha* [36] is used. As an assertion library *Chai* has been used. Together these libraries allow developers to easily write readable tests for an asynchronous code.

To simulate an environment that stores transactions in the blockchain a mock adapter is used. It stores transactions in local machine's memory, thus the code can be tested locally and it does not need to be installed on a blockchain network. The adapter also simulates the caller's identity and because of that, a developer can easily change the identity of a caller. In figure 5.5, an initialisation of the adapter can be seen. The mock adapter must be initialised with all the controllers that are used in the testing. *Participant* and *task* controllers have to reference the same adapter (lines 14 and 15) in order to access the same mocked blockchain.

```
1   adapter = new MockControllerAdapter();
2   await adapter.init([
3     {
4       version: '*',
5       controller: 'TaskController',
6       name: join(__dirname, '..')
7     },
8     {
9       version: '*',
10      controller: 'ParticipantController',
11      name: join(__dirname, '../../participant-cc')
12    }
13  ]);
14  taskManagerCtrl = ClientFactory(TaskController, adapter);
15  participantCtrl = ClientFactory(ParticipantController, adapter);
```

FIGURE 5.5: Blockchain mock adapter initialization.

To test the implemented chaincode, 21 unit tests have been created. The tests check for changes in the blockchain state when the methods are being invoked, and they also check if errors are returned when a caller invokes a method with invalid arguments. An example of a test that checks that the task has been marked as completed after it has been approved by the task owner can be seen in figure 5.6. Line 2 shows how to mock an identity of a participant.

```
    it('should mark task as completed', async () => {
        (adapter.stub as any).usercert = taskOwner;
        await taskManagerCtrl.approve(idCreatedTask);
        let retrivedTask = await adapter.getById<Task>(idCreatedTask);
        chai.expect(retrivedTask.state).to.equal(TaskState.COMPLETED);
    });
```

FIGURE 5.6: Test that checks if approve method works as expected.

5.4 Project folder description

To help the reader navigate in the created project, this section describes the project structure. The most important files are highlighted in figure 5.7. To run the individual parts of the project the reader should read a *README.md* file, where the most important commands are stated. The project is enclosed to the thesis but it is also accessible on GitHub.

5.5 Future work

To finalise the system and make it usable and deployable in real-life environment, there are several things that have to be completed.

To provide an actual overview of tasks present in the system, the front-end part of the system needs to be implemented. The front-end will receive the data from the implemented Node.js application by sending requests to its REST API. The front-end must have an intuitive user interface that will allow the organisation to have a clear overview of the tasks and will complement the efficient project management in the organisation. Once all of the system parts has been developed, an organisation can start with integration testing.

Furthermore, as previously mentioned, it is desirable to be able to attach files to the tasks. The Node.js application can be expanded in such way that, when the front-end tries to attach a file to the task, the application makes a hash of the file and stores it into the blockchain. The actual file is then saved in a different storage. It could be a traditional database or an IPFS [29].

Finally, if an organisation wants to use the proposed solution, it needs to set up the entire network of nodes, specify the consensus policy (e.g. how many nodes are needed to validate a transaction) and manage digital certificates that are provided to the users.

```
tasknet-convector/
├── packages/ ............................Folder with the
│   │                                     chaincode and the Node.js
│   │                                     application.
│   ├── participant-cc/ ..................Folder with participant
│   │   │                                 code logic.
│   │   ├── src/ .........................Folder with source files.
│   │   │   ├── participant.controller.ts ....File with participant
│   │   │   │                             controller and its main
│   │   │   │                             business logic.
│   │   │   └── participant.model.ts .........File with model definition
│   │   │                                 of a participant class.
│   │   └── package.json .................JSON file with all of the
│   │                                     dependencies.
│   ├── task-cc/ .........................Folder with task code
│   │                                     logic.  Structure is
│   │                                     similar to the participant
│   │                                     folder.
│   └── taskManager-app/ .................Folder with Node.js
│                                         application.
├── api.json .............................JSON file to generate
│                                         basic structure of
│                                         taskManager-app .
├── org1.taskManager.config.json .........File for Convector
│                                         tool.  Used to specify
│                                         the location of
│                                         Hyperledger Fabric and
│                                         needed cryptographic
│                                         material. taskManager-app .
├── package.json .........................JSON file with defined
│                                         scripts to test, install
│                                         and run chaincode.
├── tsconfig.json ........................File for setting up rules
│                                         for Typescript compilation.
└── update-paths.js ......................JavaScript file used
                                          to update paths to
                                          Hyperledger.
```

FIGURE 5.7: Project structure

Chapter 6

Conclusion

The thesis explored the suitability of using a blockchain technology for a task management domain. Analysis concluded that the domain is suitable and makes sense only in a case where individual parties using a task management software cannot be fully trusted. By utilizing blockchain technology the parties using a task management system do not need to provide any data to an external party that would need to vouch for the data integrity. In the situation when a company does not have an issue with trust and availability when using the software, the system can be implemented with traditional technologies (e.g. traditional database) or a third-party software.

For companies that do not want to provide data to a third-party, the thesis proposes solutions for how blockchain could be incorporated into a task management software, making the system distributed and tamper-proof. The first solution is based on permissionless blockchain, which enables any individual to verify the data integrity, encouraging high transparency of the system. The second solution involves permissionless blockchain, where the users of the system are chosen by the company governing the system. This solution is more suitable for private organisations. Blockchain in both solutions serve as a distributed database that stores data of only limited size, with larger data being stored off the chain and keeping only their hash inside the blockchain.

For the practical part of this thesis, a system based on permissioned blockchain has been designed and implemented. The implementation part is built upon a Hyperledger Fabric platform. Delivered implementation contains tested chaincode (smart contract) and a Node.js application with interactive REST API that communicates with the chaincode.

Bibliography

[1] Satoshi Nakamoto. *Bitcoin: A peer-to-peer electronic cash system*. URL: https://bitcoin.org/bitcoin.pdf (visited on 04/20/2019).

[2] Don Tapscott and Alex Tapscott. *Blockchain Revolution: How the Technology Behind Bitcoin Is Changing Money, Business, and the World*. Portfolio, 2016. ISBN: XXXXXX-XXXX, 9781101980132.

[3] Maher Alharby and Aad van Moorsel. "Blockchain-based Smart Contracts: A Systematic Mapping Study". In: *CoRR* abs/1710.06372 (2017). arXiv: 1710.06372. URL: http://arxiv.org/abs/1710.06372.

[4] Arvind Narayanan and Jeremy Clark. "Bitcoin's Academic Pedigree". In: *Commun. ACM* 60.12 (Nov. 2017), pp. 36–45. ISSN: 0001-0782. DOI: 10.1145/3132259. URL: http://doi.acm.org/10.1145/3132259.

[5] B. A. Scriber. "A Framework for Determining Blockchain Applicability". In: *IEEE Software* 35.4 (2018), pp. 70–77. ISSN: 0740-7459. DOI: 10.1109/MS.2018.2801552.

[6] techopedia. *Task Management*. URL: https://www.techopedia.com/definition/9652/task-management (visited on 05/25/2019).

[7] Uwe V. Riss, Alan Rickayzen, and Heiko Maus. "Challenges for business process and task management". In: *Proceedings of the 5th International Conference on Knowledge Management IKNOW '05*. 2005.

[8] American Heritage. *American Heritage® Dictionary of the English Language*. URL: https://www.thefreedictionary.com/cryptography (visited on 04/10/2019).

[9] Paul C. van Oorschot Scott A. Vanstone Jonathan Katz Alfred J. Menezes. *Handbook of Applied Cryptography*. 1997. ISBN: 9780429881329.

[10] Dawn M. Turner. *Major standards and compliance of digital signatures - a world-wide consideration*. URL: https://www.cryptomathic.com/news-events/blog/major-standards-and-compliance-of-digital-signatures-a-world-wide-consideration (visited on 04/11/2019).

[11] DocuSign. *Understanding digital signatures*. URL: https://www.docusign.com/how-it-works/electronic-signature/digital-signature/digital-signature-faq (visited on 04/15/2019).

[12] A. Narayanan et al. *Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction*. Princeton University Press, 2016. ISBN: 9781400884155. URL: https://books.google.cz/books?id=LchFDAAAQBAJ.

[13] Zerocash. *Q&A*. URL: http://zerocash-project.org/q_and_a (visited on 04/25/2019).

[14] Linux Foundation. *Hyperledger Fabric*. URL: https://www.hyperledger.org/projects/fabric (visited on 05/20/2019).

[15] R3. *Corda*. URL: https://www.r3.com (visited on 04/25/2019).

[16] Miguel Castro and Barbara Loskov. *Practical Byzantine Fault Tolerance*. 1999.

[17] Gideon Greenspan. *Avoiding the pointless blockchain project*. 2015. URL: `https://www.multichain.com/blog/2015/11/avoiding-pointless-blockchain-project/` (visited on 04/11/2019).

[18] Karl Wüst and Arthur Gervais. "Do you Need a Blockchain?" In: *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)* (2017), pp. 45–54.

[19] Blockchain Hub. *Smart contracts*. URL: `https://blockchainhub.net/smart-contracts` (visited on 05/09/2019).

[20] Aran Davies. *5 Best Smart Contract Platforms for 2019*. 2019. URL: `https://www.devteam.space/blog/5-best-smart-contract-platforms-for-2019/` (visited on 05/09/2019).

[21] Ethereum.org. *Ethereum Development Tutorial*. 2019. URL: `https://github.com/ethereum/wiki/wiki/Ethereum-Development-Tutorial#gas` (visited on 05/09/2019).

[22] monday. *monday.com*. URL: `https://www.capterra.com/p/147657/monday-com/` (visited on 04/20/2019).

[23] *Wrike*. URL: `https://www.wrike.com` (visited on 04/25/2019).

[24] Capterra. *Task Management Software*. URL: `https://www.capterra.com/task-management-software/` (visited on 05/25/2019).

[25] Dan Price. *What Is a 51 Percent Attack?* URL: `https://blocksdecoded.com/51-percent-attack/` (visited on 05/30/2019).

[26] Hasib Anwar. *Hyperledger vs Ethereum*. 2019. URL: `https://101blockchains.com/hyperledger-vs-ethereum-2/` (visited on 05/09/2019).

[27] Gavin Wood et al. "Ethereum: A secure decentralised generalised transaction ledger". In: *Ethereum project yellow paper* 151 (2014), pp. 1–32.

[28] *ETH Gas station*. URL: `https://www.ethgasstation.info/index.php` (visited on 05/03/2019).

[29] *IPFS*. URL: `https://ipfs.io` (visited on 05/01/2019).

[30] uml diagrams.org. *Use Case diagram*. URL: `https://www.uml-diagrams.org/use-case-diagrams.html` (visited on 05/11/2019).

[31] *Hyperledger Fabric documentation*. URL: `https://hyperledger-fabric.readthedocs.io/en/latest/` (visited on 05/20/2019).

[32] *Hyperledger Composer*. URL: `https://hyperledger.github.io/composer/latest/index.html` (visited on 05/23/2019).

[33] Woldsibu. *Convector*. URL: `https://github.com/worldsibu/convector` (visited on 05/24/2019).

[34] Woldsibu. *Convector identity patterns*. URL: `https://github.com/worldsibu/convector-identity-patterns` (visited on 05/25/2019).

[35] Woldsibu. *convector-rest-api*. URL: `https://github.com/worldsibu/convector-rest-api` (visited on 05/26/2019).

[36] *Mocha*. URL: `https://mochajs.org/` (visited on 05/26/2019).