Management and Analysis Platform for Data Based in Blockchain Technology

Hugo André Ferreira da Silva

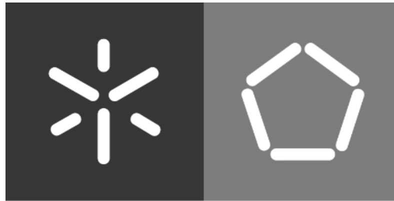
UMinho | 2021

Universidade do Minho
Escola de Engenharia

Hugo André Ferreira da Silva

Management and Analysis Platform for Data Based in Blockchain Technology

July 2021



Universidade do Minho
Escola de Engenharia

Hugo André Ferreira da Silva

Management and Analysis Platform for Data Based in Blockchain Technology

Master Dissertation
Integrated Master in Engineering and Management of Information Systems

Work done on the orientation of

Professor Doutor Manuel Filipe Vieira Torres dos Santos
Professor Tiago André Saraiva Guimarães

July de 2021

Direitos de autor e condições de utilização do trabalho por terceiros

Este é um trabalho académico que pode ser utilizado por terceiros desde que respeitadas as regras e boas práticas internacionalmente aceites, no que concerne aos direitos de autor e direitos conexos.

Assim, o presente trabalho pode ser utilizado nos termos previstos na licença abaixo indicada.

Caso o utilizador necessite de permissão para poder fazer um uso do trabalho em condições não previstas no licenciamento indicado, deverá contactar o autor, através do RepositóriUM da Universidade do Minho.

Licença concedida aos utilizadores deste trabalho

Atribuição-NãoComercial-SemDerivações
CC BY-NC-ND

https://creativecommons.org/licenses/by-nc-nd/4.0/

Acknowledgments

Throughout the writing of this dissertation, I have received a great deal of support and assistance from a group of people that I would like to mention.

I would first like to thank my supervisors, Professor Manuel Filipe Santos and Professor Tiago Guimarães for giving me the opportunity to investigate blockchain technology.

I would also like to thank both my mother, father, and sister for their unconditional support and motivation.

In addition, I would like to thank both my grandfather and grandmother for motivating me to pursue and finish this dissertation.

Lastly, I want to thank all my friends who helped me during the most complicated phases.

Statement of Integrity

I hereby declare having conducted this academic work with integrity. I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration.

I further declare that I have fully acknowledged the Code of Ethical Conduct of the University of Minho.

Abstract

Managing and sharing sensitive clinical data requires special attention and a specific set of rules that ensure its authenticity, privacy, and security. Currently, many healthcare systems have this problem, in the sense that they do not ensure data immutability. This project aims to develop a solution that not only combines the mandatory guarantee of data immutability and veracity, but also a controlled and secure way of querying data. In short, this topic consists of a software development project based on blockchain technology that aims to provide a solution, which in turn aims to raise the reliability and authenticity of data in the healthcare domain. In addition, a state of art about the various concepts of Blockchain technology and its components, algorithms, and frameworks that allow its development will be described. The methodology used for this project is the *Design Science Research.*

In this research, a Blockchain Network for the Healthcare environment was implemented as well as some blockchain-related tools for benchmarking and data visualization. Hyperledger Fabric is used as the framework for blockchain development. Hyperledger Caliper and Prometheus are used for blockchain benchmarking and Blockchain Explorer is used for blockchain data visualization. Moreover, a REST API was developed to facilitate communication with the blockchain system.

Every step of the development is documented and analyzed.

Key Word: Blockchain, Healthcare, Tracking, Technology

Resumo

A gestão e disponibilização de dados muito sensíveis como os clínicos, requer uma especial atenção e um conjunto regras muito específicas que garantam a sua autenticidade, privacidade e segurança. Vários sistemas de saúde apresentam este problema, na medida em que não garantem a imutabilidade dos dados. Com este projeto, procura-se desenvolver uma solução que combine não só a obrigatoriedade de garantir a imutabilidade e veracidade dos dados clínicos como uma controlada e segura consulta dos mesmos. Concluindo, este tema consiste num projeto de Desenvolvimento de software, que pretende disponibilizar uma solução que contribua para aumentar a fiabilidade e a veracidade dos dados em contexto hospitalar com recurso a tecnologia blockchain. Posteriormente, será descrito um estado de arte sobre os vários conceitos inerentes à tecnologia blockchain e seus componentes, algoritmos e arquiteturas que permitem o seu desenvolvimento. A metodologia utilizada para este projeto é a *Design Science Research*.

Nesta investigação, foi implementada uma Blockchain Network para a área da Saúde, bem como algumas ferramentas relacionadas com a Blockchain para o benchmarking e visualização de dados. O Hyperledger Fabric é utilizado como estrutura para o desenvolvimento da blockchain. O Hyperledger Caliper e Prometheus são utilizados para o benchmarking da blockchain e o Blockchain Explorer é utilizado para a visualização de dados. Consequentemente, uma API REST foi desenvolvida para facilitar a comunicação com o sistema blockchain.

Cada passo do desenvolvimento está documentado e analisado.

Palavras Chave: Blockchain, Saúde, Tracking, Tecnologia

Table of Contents

Acknowledgments i

Statement of Integrity ii

Abstract iii

Resumo v

Index of Tables ix

Index of Figures ix

Acronyms xii

1. Introduction 1

1.1. Document Structure 1

2. Objectives 2

3. State of Art 3

3.1. Information Systems in Healthcare 3

3.2. Blockchain 4

3.2.1. Introduction 4

3.2.2. Blockchain Structure 11

3.2.3. Public Blockchain – Permissionless 13

3.2.4. Private Blockchain – Permissioned 14

3.2.5. Private Blockchain vs Public Blockchain 15

3.2.6. Consensus Algorithms 16

3.3. Blockchain in Healthcare 19

3.4. Frameworks that support Blockchain development 21

3.4.1. Hyperledger Fabric 21

3.4.2. Hyperledger Composer 22

3.4.3. Hyperledger Convector 22

3.5. Tools for Blockchain Benchmarking 23

3.5.1. Hyperledger Caliper 23
3.5.2. Blockbench 24
3.5.3. Prometheus 25
4. Research Methodologies 27
4.1. Design Science Research 27
5. Project Development 30
5.1. Tools and Frameworks used 30
5.1.1. Hypeledger Fabric 31
5.1.2. Hyperledger Caliper 31
5.1.3. Prometheus and Grafana 31
5.1.4. Blockchain Explorer 31
5.1.5. Visual Studio Code 32
5.1.6. Go Language 32
5.1.7. NodeJS 32
5.1.8. Postman 32
5.2. Prerequisites 33
6. Results and Discussion 37
6.1. Network Structure 37
6.2. Hyperledger Fabric 38
6.3. REST API 48
6.3.1. API Architecture 48
6.3.2. Register User and Authentication Token 49
6.3.3. Create Beacon (createBeacon) 51
6.3.4. Create Doctor (createMedico) 52
6.3.5. Create Patient (createDoente) 53
6.3.6. Create Medical Device (createDispMedico) 54

6.3.7. Change Beacon room property (changeBeaconSala) 55

6.3.8. Change Patient room property (changeDoenteSala) 56

6.3.9. Change Medical Device room property(changeDispMedSala) 57

6.3.10. Change Medical Device doctor property (changeDispMedMedico) 58

6.3.11. Change Medical Device patient property (changeDispMedDoente) 59

6.3.12. Get History of Asset (getHistoryForAsset) 60

6.4. Blockchain Explorer 63

6.5. Hyperledger Caliper 67

6.6. Prometheus and Grafana 70

7. Benchmarking 73

8. Conclusion 82

9. References 84

Index of Tables

Table 1 - A comparison of popular blockchain consensus mechanisms (Baliga, 2017) . 19

Table 2 - API structure table diagram ... 42

Table 3 - Functions for asset creation and asset state change ... 43

Table 4 - API requests ... 48

Index of Figures

Figure 1 - Transaction through an intermediary vs. peer-to-peer transaction (Singhal et al., 2018) ... 5

Figure 2 - Various layers of blockchain (Singhal et al., 2018) ... 7

Figure 3 - Block attributes in a Blockchain system (Mohanta et al., 2019) ... 11

Figure 4 - Blocks in a blockchain linked through hash pointers (Singhal et al., 2018) ... 12

Figure 5 - Hashes not matching if one hash is altered (Singhal et al., 2018) ... 12

Figure 6 - Simplified permissionless blockchain architecture (Lin & Liao, 2017) ... 13

Figure 7 - Simplified permissioned blockchain architecture (Lin & Liao, 2017) ... 14

Figure 8 - PBFT consensus approach (Singhal et al., 2018) ... 18

Figure 9 - Hyperledger Caliper Architecture, retrieved from (Hyperledger Caliper Architecture, n.d.) ... 24

Figure 10 - Architecture of Blockbench (Wang et al., 2019) ... 25

Figure 11 - Prometheus Architecture (Prometheus, 2019) ... 27

Figure 12 - Design Science Research diagram (Peffers et al., 2008) ... 30

Figure 13 - Peer command execution in order to validate the installation ... 37

Figure 14 - Blockchain Network Architecture ... 38

Figure 15 - Chaincode Lifecycle ... 41

Figure 16 - Smart Contract structs ... 44

Figure 17 - Smart Contract function createDispMedico ... 44

Figure 18 - Smart Contract function changeDispMedDoente ... 45

Figure 19 - Smart Contract function getHistoryForAsset ... 46

Figure 20 - Smart Contract Invoke method ... 47

Figure 21 - Create a User ... 50

Figure 22 - Bearer Token being defined in the Create Beacon function 50
Figure 23 - createBeacon function API call 51
Figure 24 - createMedico function API call 52
Figure 25 - Medico_08 asset in the database 53
Figure 26 - createDoente function API call 53
Figure 27 - createDispMedico function API call 54
Figure 28 - changeBeaconSala function API call 55
Figure 29 - Beacon_10 asset in the database 56
Figure 30 - changeDoenteSala function API call 57
Figure 31 - Doente_11 asset in the database 57
Figure 32 - changeDispMedSala function API call 58
Figure 33 - changeDispMedMedico function API call 59
Figure 34 - changeDispMedDoente function API call 60
Figure 35 - Dispositivo_07 asset in the database 60
Figure 36 - getHistoryForAsset function API call 61
Figure 37 - Detailed view of the getHistoryForAsset response body for the Dispositivo_07 asset 62
Figure 38 - Blockchain Explorer Login screen 64
Figure 39 - Blockchain Explorer Dashboard 65
Figure 40 - Blockchain Explorer Transaction details 66
Figure 41 - Blockchain Explorer Block details 66
Figure 42 - Blockchain Explorer Chaincode details 66
Figure 43 - Blockchain Explorer channels 66
Figure 44 - Blockchain Explorer network participants 67
Figure 45 - Hyperledger Caliper CreateDispMedico function for random asset generation 68
Figure 46 - Hyperledger Caliper report 70
Figure 47 - Prometheus folder structure 71
Figure 48 - Prometheus 72
Figure 49 - Grafana 73
Figure 50 - Performance Metrics (1k transactions per function) 74
Figure 51 - Performance Metrics (10k transactiosn per function) 74

Figure 52- Average Send Rate vs Throughput 75
Figure 53 - Average Failed Transactions 76
Figure 54 - Max, Min and Average Latency for 100, 1k and 10K TPS 76
Figure 55 - Caliper container CPU usage during a benchmark 77
Figure 56 - Caliper container memory usage during a benchmark 78
Figure 57 - All blockchain related containers accumulated memory usage 78
Figure 58 - All blockchain related containers accumulated CPU usage 79
Figure 59 - Peer 1 Org 1 memory usage 80
Figure 60 - Peer 1 Org 1 CPU usage 80
Figure 61 - Peer 0 Org 1 memory usage 80
Figure 62 - Peer 0 Org 1 CPU usage 81

Acronyms

API – Application Programming Interface

CA – Certificate Authority

CPU – Central Processing Unit.

HTTP – Hypertext Transfer Protocol

IS – Information Systems

P2P – Peer to Peer

PBFT – Practical Byzantine Fault Tolerance

PoC – Proof of Concepts

PoS – Proof of Stake

PoW – Proof of Work

REST – Representational State Transfer

SUT – System Under Test

URL – Uniform Resource Locator

1. Introduction

Managing and sharing sensitive clinical data requires special attention and a specific set of rules that ensure its authenticity, privacy, and security. Currently, many healthcare systems are lacking in these aspects. Blockchain technology provides a solution for this problem as it guarantees a chronological order of data as well as ensuring its authenticity, privacy, and security.

This project aims to develop a management and analysis platform for data based in blockchain technology, that not only combines the mandatory guarantee of data immutability and veracity, but also a controlled and secure way of querying data.

1.1. Document Structure

The first chapter includes a brief introduction to the dissertation topic, as well as the motivation behind the project.

The second chapter contains the dissertation objectives, as well as the investigation question.

The third chapter contains the state of art, where relevant concepts about the dissertation topic are accounted for.

The fourth chapter includes the research methodologies used in this project.

The fifth chapter comprehends the development phase of the project. All the tools, frameworks, and software are specified in this chapter, as well as the prerequisites necessary in order to develop the solution.

The sixth chapter comprehends all the results obtained during the project.

Lastly, the seventh chapter contains the benchmarks and the respective graph analysis.

2. Objectives

This chapter comprehends all the macro and micro objectives associated with this work.

This project aims to develop a solution that helps to improve healthcare data privacy, veracity and reliability, by deploying a tamper-proof and immutable way of storing data. From this assumption, the following investigation question was formulated:

In which way can Blockchain Technology be used, in the Healthcare context, to develop a solution that ensures a tamper-proof, immutable, controlled, and secured way of storing data in order to achieve healthcare data veracity, privacy, and reliability?

1. Understand the requirements and necessities of the project;
2. Study the benefits and difficulties associated with the use of this type of technologies;
3. Understand the state of the art of blockchain solutions and architectures;
4. Study and comprehend the specific necessities that this type of solution raises;
5. Propose an architecture that provides a solution to the necessities and requirements of the project;
6. Develop a functional prototype featuring Web API's and Blockchain.

3. State of Art

This chapter addresses Information Systems and their relations to Healthcare, as well as come concepts about the Blockchain technology and its components, algorithms, tools for benchmarking, and frameworks that allow its development. Moreover, it will be described how blockchain relates to healthcare and some benefits of blockchain's use in healthcare will be pointed out.

3.1. Information Systems in Healthcare

Healthcare has an effect on our quality of life and how we work in society. Mistakes have serious consequences that can affect our ability to carry out social and productive endeavors. These errors are expensive, increase the duration of a patient's stay in the hospital, and cost human lives. Therefore, healthcare quality is diligently pursued and vigilantly executed, and IS can facilitate such a goal by highlighting and tracking errors at different stages in the process of care.

Another aspect of healthcare information is that it is sensitive. As a consequence, any information transfer between parties through technology is fraught with danger. For example, the data might end up in the wrong hands. Electronic storage is perceived as having a higher likelihood of leakage, thus patients' perception of the probability of compromised privacy is often higher than the actual probability. Therefore, it's critical to comprehend the contextual aspects that influence people's willingness to provide medical information in an electronic format.

One of the obstacles to healthcare technology adoption is that influential players in the industry often oppose it. Part of this arises from professional norms. They are more concerned with treating the patient and disregard other activities.

The tension between the need for orderly routines and the need for flexibility to change in local circumstances exists in the healthcare delivery environment. This tension magnifies both the complexity and importance of effective learning and adaptation surrounding healthcare IS implementation and use. For instance, systems and implementation techniques that work well in one setting may fail in another.

From numerous points of view, learning and adjustment are two of a kind when it comes to new IS. Learning is required to decide the best approach to adjust both technology and organization to accomplish a solid match between the abilities the innovation manages and the ideal examples of real use. When the required adjustments have been recognized, a distinctive sort

of learning is required to fuse these adjustments into organizational routines and to guarantee nonstop improvement. (Fichman et al., 2011).

According to (Victor, 2013), the General Data Protection Regulation proposes a range of new individual rights designed to protect consumers whose personal information is collected, processed, and stored by corporations and other entities. It establishes a consumer's "right to be forgotten" where entities that collect or process data must delete any data related to an individual.

3.2. Blockchain

This chapter talks about blockchain technology and every concept related to it, such as the various types of blockchain and algorithms that are part of its structure.

3.2.1. Introduction

According to (Crosby et al., 2016), a blockchain is a decentralized database of records, also known as a public ledger, that records transactions or digital events that participants exchange. Each transaction must be verified by the consensus of most participants in the ledger. Moreover, it's relevant to note that the blockchain contains a verifiable record of all the transactions made. To demonstrate why blockchain provides a secure environment, we can use a basic analogy such as "it is easier to steal a cookie from a cookie jar, kept in a secluded place, than stealing the cookie from a cookie jar kept in a market place, being observed by thousands of people".

A blockchain is a distributed ledger that keeps a permanent, tamper-proof record of transactions. It relies on a peer-to-peer (P2P) network, hence it is considered completely decentralized. Each node of the network maintains a copy of the ledger to prevent failures. All copies are simultaneously updated and checked at the same time.

Blockchain was originally created to solve the double-spending problem in crypto-currency. However, currently, numerous projects explore blockchain applications in multiple use cases and use them as a safe and reliable way to build and manage a distributed database and keep track of all digital transactions (Hammi et al., 2018).

According to (Singhal et al., 2018), blockchain is a distributed ledger system that allows people to exchange value in a peer-to-peer fashion. It means that no trusted intermediary, such as banks, brokers, or other escrow services, is required to act as a trusted third party.

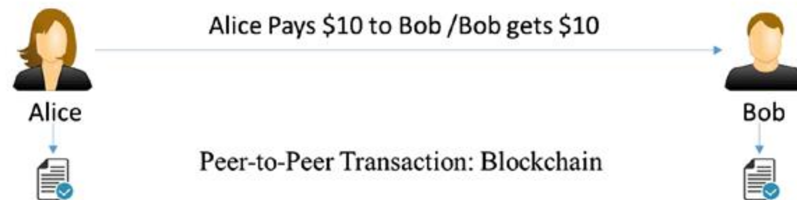


Figure 1 - Transaction through an intermediary vs. peer-to-peer transaction (Singhal et al., 2018)

There are various factors that characterize Blockchain systems. In what follows, important characteristics in the aspects of deployments, implementation, and properties will be identified (Viriyasitavat & Hoonsopon, 2019):

- Private, Public, and Permissioned Blockchain: The distinction among these types of Blockchains is the scheme of ledger sharing and who can participate in a system. Ledgers are shared and validated by a designated group of nodes in a private Blockchain. Nodes that want to be a part of the system must first be initiated or validated. Private Blockchain is best suited to closed networks in which all nodes are completely trustworthy. A public Blockchain network is completely opened and distributed. Anyone can participate or leave the system. Permissioned blockchains are a combination of private and public blockchains in which several parties are involved and the key nodes are carefully chosen at the outset. Permissioned blockchain is ideal for semi-closed networks involving a few businesses that are often structured as a consortium.
- Centralization and Decentralization: Blockchain technology is a promising solution to the distributed transaction management problems, being conducted among peers in P2P network. Public Blockchains operate in a fully decentralized environment, allowing trust of the transactions to be established among previous unknown or

untrusted nodes. To achieve the same stance, private blockchains run in a closed and trusted environment and use access management techniques.

– Persistency: Transactions recorded in a Blockchain ledger are considered persistent as they spread across the network, where each node maintains and controls its records. Several properties are derived from this characteristic including transparency, immutability, and tamper resistance.
– Validity: Unlike other distributed networks, blockchains do not require each node to execute commands. Other nodes in a blockchain system will verify transactions or blocks broadcasted. Any type of falsification can be detected easily.
– Anonymity and Identity: The key feature of public blockchains is anonymity. This system's identity is not tied to a user's real-world identity. To avoid identity theft, a single user may create multiple identities. There is no need for any central entity to maintain private information. As a consequence, based on transaction details, a real-world identity cannot be determined, retaining some privacy. On the other hand, in settings such as private and permissioned blockchains, identification is normally necessary for systems that are run and controlled by established entities.
– Auditability: The use of a recorded timestamp and persistent data allows for simple verification and tracing of previous records through nodes in a Blockchain network. Since nodes are managed, private blockchains are the least auditable. Permissioned Blockchains, which are used by certain entities, come in second, because, some agreements, such as encrypted data, can make it impossible for the information to be fully accessed. Blockchains that are auditable and public rank highest because nodes are genuinely distributed.
– Closedness and Openness: Public Blockchains rely on public nodes to maintain records of transactions. Permissioned Blockchains are considered semi-opened as nodes are pre-specified or validated before joining. The information in this blockchain is governed by the consortium's policies, which can regulate whether the information is completely accessible, partially open, or locked. Private blockchains, including permissioned blockchains, use policies to govern how nodes are chosen and the degree of data transparency. However, they rely on a single entity or owner.

According to (Baliga, 2017), when looking for Blockchain to solve a determined business problem, it is very important to look at the scale of the intended network, the relationships between participants, and both functional and non-functional aspects before determining the right platform and the right consensus model to use.

With its core characteristics, blockchain has shown its ability to disrupt conventional industries. Anonymity and auditability are both benefits of decentralization. (Zheng et al., 2017).

According to(Singhal et al., 2018), there are five high-level layers in a blockchain. These layers were formulated to gain a deeper understanding of the technology and provide a comparative analogy between the hundreds of blockchain/cryptocurrency variants available on the market. The following image (Fig. 2) shows the five layers of blockchain.

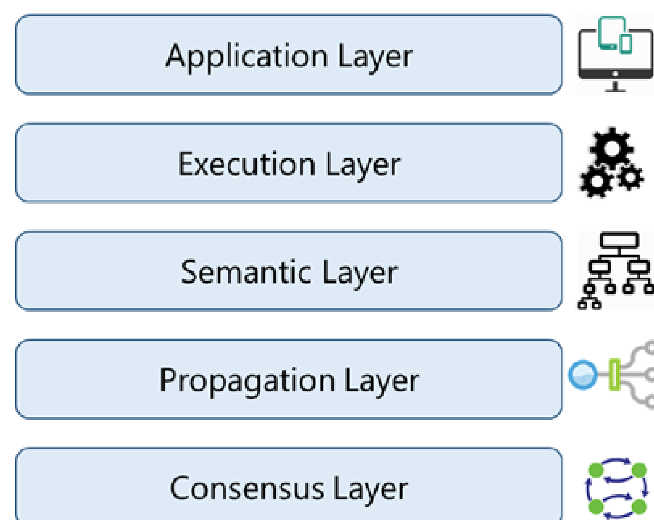


Figure 2 - Various layers of blockchain (Singhal et al., 2018)

- **Application Layer:** This is the layer where you code the desired functionalities and transform them into a user-friendly program. Client-side programming constructs, scripting, APIs, development frameworks, and other popular software development technologies are commonly used. To put it another way, this definition ensures that the heavy lifting is performed at the application layer, or that bulky storage requirements are handled off the chain, allowing the main blockchain to be light and efficient while network traffic is kept to a minimum.
- **Execution Layer:** The Execution Layer is where all of the nodes in a blockchain network execute the instructions ordered by the Application Layer. Simple instructions

or a collection of multiple instructions in the form of a smart contract may be used. In this case, a program or a script must be run to ensure that the transaction is completed correctly. The programs/scripts must be executed independently by each node in a blockchain network. Deterministic execution of programs/scripts on the same set of inputs and conditions consistently generates the same output across all nodes, reducing inconsistencies.

- **Semantic Layer:** Since the transactions and blocks are ordered, the Semantic Layer is a logical layer. A transaction has a series of instructions that pass through the Execution Layer but are checked in the Semantic Layer, whether true or invalid. The system's laws, such as data models and structures, can be described in this layer. There may be circumstances that are more complicated than simple transactions. Smart contracts also include complex instruction sets. When a smart contract is invoked in response to a transaction, the system's state is changed. A smart contract is a form of account that contains executable code as well as private states. A block typically includes a number of transactions as well as some smart contracts. The semantic layer specifies how the blocks are connected to one another. Any block in a blockchain, all the way to the genesis block, contains the hash of the previous block.
- **Propagation Layer:** The Propagation Layer is a peer-to-peer communication layer that enables nodes to discover one another, as well as communicate and sync with one another about the current state of the network. We know that when a transaction is completed, it is transmitted to the entire network. When a node proposes a valid block, it is automatically propagated to the entire network, allowing other nodes to expand on it and render it the most recent block. As a result, transaction/block propagation in the network is specified in this layer, which ensures network stability. Latency problems for transaction or block propagation are common in the asynchronous Internet network. Depending on the power of the nodes, network bandwidth, and a few other variables, some propagations happen in seconds and others take longer.
- **Consensus Layer:** For most blockchain systems, the Consensus Layer serves as the foundation. The primary aim of this layer is to get all the nodes to agree on one clear state of the ledger. Depending on the use case, different methods for achieving consensus among the nodes can exist. This layer ensures that the blockchain is safe

and secure. In order for a decentralized blockchain to be self-sustaining, it must have some kind of incentive system that not only keeps the network alive but also enforces consensus. To choose a node that can propose a block, Bitcoin and Ethereum use a Proof of Work (PoW) consensus method. There are several different variants of consensus protocols such as Proof of Stake (PoS), deligated PoS s(dPoS), Practical Byzantine Fault Tolerance (PBFT).

According to (Singhal et al., 2018), the advantages of decentralized systems like blockchain systems over centralized systems are the following:

- Elimination of intermediaries
- Easier and genuine verification of transactions
- Increased security with lower cost
- Greater transparency
- Decentralized and immutable

Regarding blockchain adoption, different flavors of blockchain offerings, such as Ethereum and Hyperledger, have been developed by some businesses. For instance, on their Azure and Bluemix cloud platforms, Microsoft and IBM have established SaaS (Software as a Service) offerings. Various startups were created, and several existing businesses adopted blockchain projects aimed at solving certain business problems that had previously been difficult to solve. It has had a major influence on the financial services industry. It's difficult to think of a major bank or financial institution that isn't looking into blockchain Aside from the stock industry, projects are now underway in fields such as media and entertainment, oil trading, prediction markets, retail chains, loyalty incentive programs, and more insurance, distribution, and supply chains, as well as medical records applications in the government and military. There are still some technological difficulties. Blockchain is still in its early stages, thus mainstream adoption may take a few years longer. There are currently several proposals to solve the scalability problems with blockchain (Singhal et al., 2018).

According to (Singhal et al., 2018), the decentralized architecture of blockchain defies the centralized design. The distinction between decentralized and centralized isn't always apparent. They are often misunderstood and poorly described. The explanation for this is that there is almost

no strictly centralized or decentralized structure. Consequently, three perspectives characterize whether a system is centralizer or decentralized:

- **Technical Architecture:** From the standpoint of technological architecture, a system may be centralized or decentralized. We take into account how many physical computers (or nodes) are used to build a system, how many node failures it can withstand before the entire system fails, and so on.
- **Political perspective:** This viewpoint describes how much power a person, a group of people, or an entire organization has over a system. If they have power over the system's computers, the system is necessarily centralized. In a political context, though, if no individual person or party controls the system and everyone has equal access to it, it is a decentralized system.
- **Logical perspective:** Regardless of whether a structure is theoretically or politically centralized or decentralized, it may be logically centralized or decentralized depending on how it looks. Another example is that if you vertically split a system in half, with each half having service providers and customers, they are decentralized if they can run as individual units and centralized if they can't.

All of the above viewpoints are important when developing a real-world system and determining whether it is centralized or decentralized. The following examples help to clarify the concept (Singhal et al., 2018):

- Corporates are architecturally centralized (one headquarters), politically centralized (governed by a CEO or board of directors), and theoretically centralized. (It's impossible to cut them in half.)
- Our communication language is decentralized from every viewpoint, architecturally, politically, and logically. In general, when two people interact with each other, their language is neither politically influenced nor logically dependent on the language of other people's communication.
- BitTorrent and other torrent networks are also decentralized in almost any way. Since any node can be a provider or a customer, the device can be cut in half and still work.

- The Content Delivery Network, on the other hand, is architecturally and technically decentralized, but it is politically centralized due to corporate ownership. Amazon CloudFront is an example.
- Let's take a look at blockchain right now. Blockchain aimed to allow for decentralization. As a result, it is designed to be architecturally decentralized. From a political standpoint, it is also decentralized and no one owns it. It is, however, technically centralized since there is a single agreed-upon state and the whole system acts as a single global device.

3.2.2. Blockchain Structure

According to (Mohanta et al., 2019), a block is a list of valid transactions in the Blockchain. Any node in a Blockchain system may initiate a transaction, which is broadcast to all other nodes in the network. The transaction is checked by network nodes using previous transactions, and then the transaction is added to the current Blockchain. The following image (Fig 3) represents the block structure present in the blockchain.

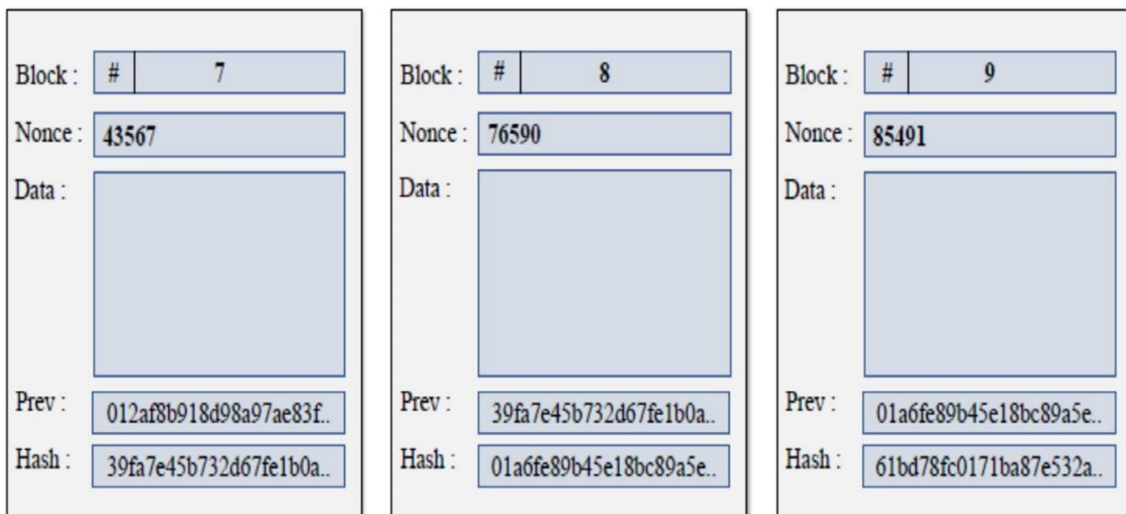


Figure 3 - Block attributes in a Blockchain system (Mohanta et al., 2019)

Every block is a descendant of the one before it. The Blockchain system uses the hash of previous blocks to generate the hash of new blocks, making it tamper-proof (Mohanta et al., 2019).

In the sense that it is a chain of blocks linked together, a blockchain is actually a blockchain data structure.

The basic building block of the blockchain data structure is hash pointers. A hash pointer refers to a cryptographic hash that points to a data block. The data block's hash is used as the hash pointer. Hash pointers refer to the previous data block and allow you to check that the data isn't tampered with. The hash pointer's goal is to create a tamper-proof blockchain that can be used as a single source of evidence. The previous block's hash is stored in the current block header, and the next block's hash, along with its block header, is stored in the header of the next block. The following image (Fig. 4) represents the blocks and the respective hash pointers (Singhal et al., 2018).

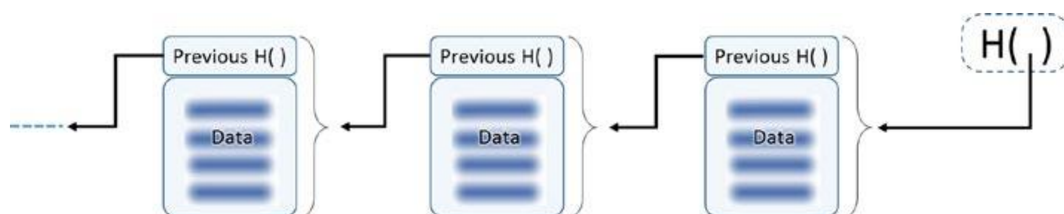


Figure 4 - Blocks in a blockchain linked through hash pointers (Singhal et al., 2018)

Every block, as can be seen, points to the block before it, referred to as "the parent block." Any new block added to the chain serves as the parent block for the subsequent blocks. It goes all the way to the "genesis block," which is the first block in the blockchain to be formed. No one can change data in any block in such a design where blocks are connected back with hashes. If the data is changed, the hashes will not match. Any attempt to change the content of the Header or Block breaks the entire chain. Assume that the data in block-1234 has been modified. If you do this, the hash stored in block-1235's block header will not correspond. This can be visualized in the image below (Fig. 5) (Singhal et al., 2018).

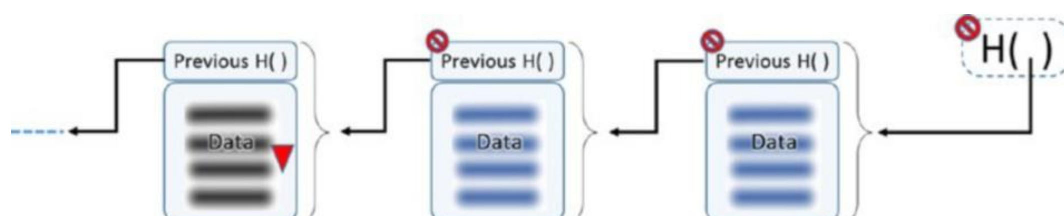


Figure 5 - Hashes not matching if one hash is altered (Singhal et al., 2018)

This means that in order to alter the hash of one block you need to alter the subsequent blocks. This must be done all the way through to the final or most recent hash. It is impossible to hack into the majority of the networks and alter all the hashes at once because several others in the network already have a copy of the blockchain and the most recent hash (Singhal et al., 2018).

3.2.3. Public Blockchain – Permissionless

Public blockchains, also known as, Permission-less ledgers, are open to the public and allow anybody to participate as a node in the decision-making process. Users can, or not, be rewarded for their participation. The ledgers are not owned by anyone and are publicly available for everyone to participate in. Every user of the permission-less ledger maintains a copy of it on their nodes and uses a distributed consensus mechanism in order to decide about the state of the ledger. (Bashir, 2017)

According to (Lin & Liao, 2017), in public blockchains, everyone can check the transactions and verify them, and also participate in the consensus process (Fig. 6).

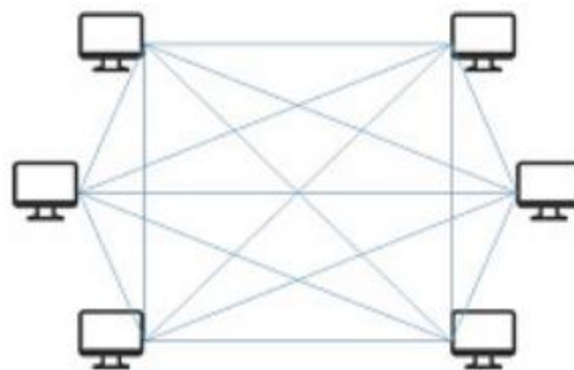


Figure 6 - Simplified permissionless blockchain architecture (Lin & Liao, 2017)

According to (Hammi et al., 2018), Bitcoin is a cryptocurrency and a digital payment system, based on a public blockchain. Each block has a list of its transactions stored in the header. Every node in the network can be a miner and stores a copy of the current blockchain. Transactions are recorded by order and have a timestamp associated. In order to validate the transactions, Bitcoin uses a consensus mechanism. To make the mechanism resistant to attacks, Bitcoin uses a PoW mechanism. This PoW represents a data processing challenge, that is costly and time-consuming, and is executed for every new block. On the other hand, it's simpler for others to verify

it. In short, when a node sends a constructed block over the network, all the recipients verify the block's transactions as well as its PoW. If most of the network nodes agree upon a block, the latter is validated and added to the blockchain. The block maker is rewarded after the remaining nodes upgrade their blockchain copies.

Theoretically, Bitcoin blocks can be falsified only if more than half of the nodes in the network are corrupted. This, however, is nearly impossible to achieve.

Etherenum is an innovative blockchain-based virtual machine that features stateful user-created digital contracts (Olleros & Zhegu, 2016).

This public blockchain provides a cryptocurrency called Ether used for paying financial transactions as well as application processing. For blocks' validation, Ethereum uses a PoW mechanism called Ethash, however, there's also a beta version of Ethereum that uses the Casper protocol, which is based on a PoS. When a miner creates a block, it sends it through the network with its PoW. Some blocks are supposed to be generated at the same time. Therefore, it keeps the first in its main chain and considers the others as *Uncles.* The chain that contains more *Uncles* is kept as the main chain at the end of consensus. (Hammi et al., 2018)

3.2.4. Private Blockchain – Permissioned

Private blockchains, as the name implies are private. This type of blockchain is open only to a consortium or group of individuals or organizations that have decided to share the ledger among themselves (Bashir, 2017)

In private blockchains, a node is restricted. Not every node can participate in the consensus. The blockchain has strict authority management on data access (Fig. 7) (Lin & Liao, 2017).

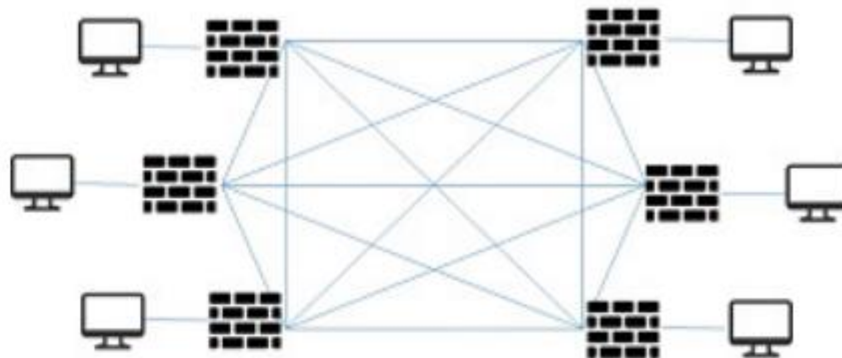


Figure 7 - Simplified permissioned blockchain architecture (Lin & Liao, 2017)

Hyperledger is among the most popular private blockchains (Dinh et al., 2018).

According to (Cachin, 2016), Hyperledger Fabric is a distributed ledger framework for running smart contracts using blockchain technology, leveraging technologies that are well-known and well-proven, with a modular architecture that allows for pluggable implementations of different functions. It is a blockchain platform aimed for business use and one of the multiple projects currently in development under the Hyperledger Project. It's open-source and standards-based, and it can run user-defined smart contracts. It's also built on a modular architecture with pluggable consensus protocols, and it has solid security and identity features.

The fabric's distributed ledger protocol is controlled by peers. The fabric distinguishes between two kinds of peers, validating peers and non-validating peers. A validating peer is responsible for validating transactions, hence its designation. It also runs consensus. On the other hand, non-validating peer functions as a proxy to connect clients, distribute transactions, to validating peers. Although a non-validating peer does not perform transactions, it may check them.

Transactions can be public or confidential, depending on the nature of the data stored. As a consensus method, Hyperledger utilizes the Practical Byzantine Fault Tolerant (PBFT). (Hammi et al., 2018).

3.2.5. Private Blockchain vs Public Blockchain

There are two types of blockchain systems: public and private. Any node can enter and exit the system in the former, making the blockchain fully decentralized and similar to a peer-to-peer system. The blockchain enforces strict membership in the latter. There is an access control process in place to decide who may enter the system. As a consequence, each node is authenticated, and the other nodes are aware of its identity. (Dinh et al., 2018).

Typically, Public Blockchains use computational or memory complexity to achieve robust consensus among a large number of untrusted peers while sacrificing transaction finality and throughput. Permissioned, Private Blockchains, on the other hand, are opting for a less scalable but far higher throughput model that ensures faster transaction completion. (Baliga, 2017).

3.2.6. Consensus Algorithms

Regarding Consensus algorithms and according to (Bashir, 2017), there are two categories:

- Proof-based, leader-based, or the *Nakamoto* consensus.
- Byzantine fault tolerance-based, which is a more traditional approach based on rounds of votes.

3.2.6.1. Proof of Work (PoW)

This type of consensus algorithm relies on proof that enough computational resources have been spent before proposing a value for acceptance in the network. This is used in Bitcoin and other cryptocurrencies (Bashir, 2017).

According to (Zheng et al., 2018), each network node calculates a hash value of the constantly changing block header in PoW. The measured value must be equal to or less than a certain threshold, according to the agreement. When one node obtains the value, all other nodes must mutually confirm the correctness of the value. Moreover, transactions in the new block are validated to prevent fraud. At that point, the assortment of transactions utilized for the computations is affirmed to be the validated outcome, which is denoted by a new block in the blockchain. The nodes that execute these computations are called miners and the process of calculating is called mining. Since this is a time-consuming process there is usually a reward associated, which is the case of Bitcoin.

According to (Singhal et al., 2018), the PoW algorithm operates by doing some work on a block of transactions before proposing it to the entire network. A Proof of Work (PoW) is a piece of data that is difficult to generate in terms of computation and time but simple to check. If any compute-intensive work needs to be done before generating a block, said work will be beneficial in two ways: first, it can take some time, and second, if a node is attempting to insert a fraudulent transaction into a block, rejection of that block by the majority of the nodes will be very expensive for the one proposing the block because the computation used to obtain the PoW will be useless. Proposing a node with a fraudulent transaction and being rejected would not have been a big deal if it was done with almost no effort. Putting in the effort to propose a block prevents a node from inadvertently injecting a fraudulent transaction. Also, the work's complexity should be adjustable so that the rate at which the blocks are generated can be controlled.

3.2.6.2. Proof of Stake (PoS)

This algorithm is based on the concept that a node has enough stake in the network (Bashir, 2017).

Proof of stake algorithms was designed to overcome the disadvantages of Proof of Work algorithms by replacing the mining operation with an alternative approach involving a user’s stake in the blockchain system. The PoS algorithm pseudo-randomly selects validators for block creation, thus ensuring that validators cannot predict their turn. (Baliga, 2017).

According to (Zheng et al., 2018), since it is assumed that people with more currencies would be less likely to target the network, PoS needs people to show ownership of the amount of currency. This may be unjust because the network's wealthiest member is inevitably powerful. When compared to PoW, PoS uses less energy and is more effective. Unfortunately, since the cost of mining is so low, attacks can occur as a result.

According to (Singhal et al., 2018), to engage in validating transactions in a PoS scheme, validators must bond their stake. A validator's chance of creating a block is proportional to the amount at stake. The higher the stake, the better their chance of validating a new block of transactions. A validator just needs to show that they own a certain percentage of all coins in a given currency system at any given time. For example, if a validator owns 2% of all Ether (ETH) in the Ethereum network, they can validate 2% of all transactions. As a result, who gets to generate the new transaction block is determined. PoS algorithms include naive PoS, delegated PoS, chain-based PoS, BFT-style PoS, and Casper PoS, among others. In comparison to PoW systems, a PoS system operates much faster because the maker of a block is deterministic, based on the sum at stake. Furthermore, since there are no block incentives and only transaction fees, all digital currencies must be generated at the outset, and their total value must remain constant throughout. Since executing an attack will endanger the entire amount at stake, PoS systems could provide better security against malicious attacks. PoS is less power-hungry than PoW, therefore the latter is less prioritized when applicable.

3.2.6.3. Byzantine Fault Tolerance (PBFT)

This algorithm was the first to give a practical solution to achieve consensus in the face of Byzantine failures. It uses a concept of a replicated state machine and voting by replicas for state changes. It also includes signing and encryption of messages exchanged between replicas and

clients. The algorithm requires “3f+1” replicas to be able to tolerate “f” failing nodes. This approach translates into a good performance. However, its messaging overhead increases significantly as the number of replicas increases. (Baliga, 2017)

According to (Zheng et al., 2018), a new block is decided in a round. The whole process can be divided into three phases: pre-prepared, prepared, and commit. In each phase, a node is eligible to go to the next phase if it has received votes from over 2/3 of all nodes thus, PBFT requires that every node is known to the network.

According to (Singhal et al., 2018), the Practical Byzantine Fault Tolerance algorithm (PBFT) is one of the many consensus algorithms that can be used in a blockchain application. Hyperledger, Stellar, and Ripple are the only blockchain projects that use PBFT consensus. Similar to PoS algorithms, PBFT is an algorithm that does not produce mining rewards, but the technicalities of their respective implementations, however, are distinct. Requests are broadcast to all participating nodes with their own replicas or internal states. When nodes receive a message, they use their internal states to perform the computation. The computation's result is then distributed to all other nodes in the system. As a result, each node is aware of what other nodes are working on. They make a decision and agree to a final value, which is again spread across the nodes, based on their own computation results as well as those obtained from the other nodes. At this time, every node is aware of all other nodes' final decisions. Then they all answer with their final decisions, and the final consensus is reached based on the majority. Based on the effort needed, PBFT can be more effective than other consensus algorithms. Nevertheless, because of the way this algorithm is constructed, the system's privacy could be adversely affected. Even in non-blockchain contexts, it is one of the most commonly used consensus algorithms.

The following image (Fig. 8) demonstrates the process.

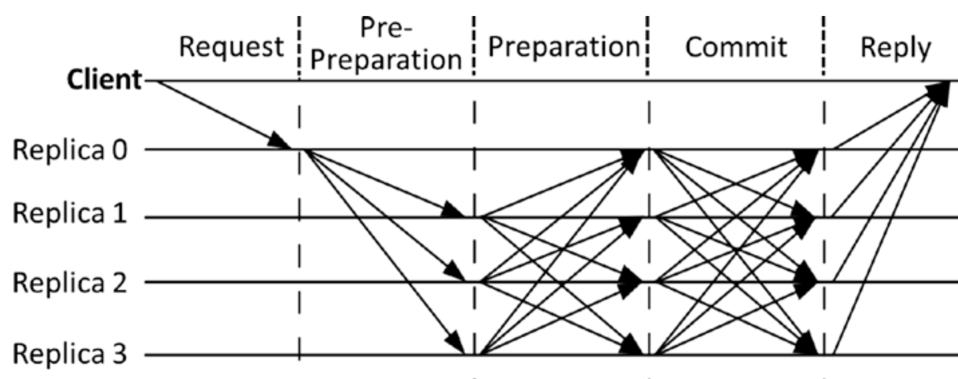


Figure 8 - PBFT consensus approach (Singhal et al., 2018)

3.2.6.4. *Comparative analysis*

In this chapter, a table (Table 1) will resume the comparative analysis between the previously explained algorithms, according to (Baliga, 2017).

Table 1 - A comparison of popular blockchain consensus mechanisms (Baliga, 2017)

	PoW	**PoS**	**PBFT**
Blockchain type	Permissionless	Both	Permissioned
Transaction finality	Probabilistic	Probabilistic	Immediate
Transaction rate	Low	High	High
Cost of participation	Yes	Yes	No
Scalability of peer network	High	High	Low
Trust model	Untrusted	Untrusted	Semi-trusted
Adversary Tolerance	<=25%	Depends on specific algorithm used	<=33%

3.3. Blockchain in Healthcare

The use of blockchains as an underlying medium for Health Information Exchange (HIE), or health transactions between patients, providers, payers, and other relevant parties, is the most talked-about among healthcare applications (Kuo et al., 2017).

Hospitals must communicate through an exchange of health information throughout the process of managing and treating patients since health interoperability is centered on the transmission of data between peers. Nevertheless, increased interoperability poses new issues and demands in terms of security and privacy, as well as technology and governance. Solving these issues, which are currently unsolved for traditional interoperability, is part of the problem (Guimarães et al., 2020).

Many research and ongoing initiatives are focusing on using blockchains to exchange patient care data in order to enhance medical record management. Another critical goal is to validate claim transactions to facilitate healthcare funding tasks such as preauthorization payment, alternative payment models, automated claims using Fast Healthcare Interoperability Resources and smart contracts, and Smart Health Profile to better control Medicaid beneficiaries' frequent exit and reentry due to eligibility adjustments. Many researchers also consider using blockchain technology and blockchain-based data exchange networks to speed up secondary uses of clinical data.

Besides exploiting blockchains as ledgers of patient care data, many studies and projects have also proposed using them to store various types of healthcare-related data, such as genomic and precision medicine data, patient-centered or patient-related outcomes data, provider/patient directories and care plans data, clinical trial data, patient consent data, pharmaceutical supply chain data, and biomarker data.

There are many benefits of blockchain implementation to advance healthcare data ledgers. The following list comprehends these benefits and the impact it has in healthcare systems (Kuo et al., 2017):

- Decentralized Management: All data is stored in a decentralized manner, with no single entity storing or having singular authority to access
- Immutable Audit Trail: Unchangeable log of clinical research protocols
- Data Provenance: Ensure original manufacturer and ownership transferring in pharmaceutical supply chain
- Robustness/Availability: Improved robustness for counterfeit drug prevention/detection systems in the pharmaceutical supply chain
- Security/Privacy: Higher patient confidence in consent recording systems since patients can add consent statements at any point in their care journey

Blockchain also supports the acceleration of healthcare research. Benefits include:

- Improved care data sharing and analysis without ceding control
- Trackable and timestamped patient-generated data
- Superior healthcare data availability

- Secured and privacy-preserving health care data sharing

However, there are so many challenges that should be considered when adopting blockchain technology in the healthcare domain.

The first challenge is related to transparency and confidentiality. In a blockchain network, everyone can see everything, so there is high transparency and low confidentiality. Open transparency of information during a transaction is usually considered a limitation of blockchain. Moreover, even if a user is anonymized by using hash values, the user may still be identified through the inspection and analysis of the publicly available transaction information on the network. This issue is critical for healthcare applications because patient-related data is highly sensitive.

The second challenge is speed and scalability. Transaction times can be long, depending on the protocol used. Therefore, a speed constraint may limit the scalability of blockchain-based applications. This issue is important when developing real-time and scalable blockchain-based healthcare applications.

The third and last challenge is the threat of a 51% attack. This attack happens when there are more malicious nodes than honest ones in the network, so the consensus is corrupted. This issue is critical for healthcare applications that must be security demanding (Kuo et al., 2017).

3.4. Frameworks that support Blockchain development

In this chapter, some frameworks that support blockchain development will be presented as well as a few key features of the latter that will be described and analyzed.

3.4.1. Hyperledger Fabric

According to (Cachin, 2016), Hyperledger Fabric is a distributed ledger platform for smart contracts that uses well-known and well-proven technologies and has a modular architecture that allows for pluggable implementations of different functions. It's one of several Hyperledger Project projects currently in development.

Validating peers execute a replicated state machine that accepts three types of transactions as operations using a Byzantine Fault Tolerance consensus protocol:

- Deploy transaction: As a parameter, it accepts a chaincode (representing a smart contract) written in Go; the chaincode is mounted on the peers and ready to use;
- Invoke transaction: Invokes a transaction of a specific chaincode that was previously installed through a deploy transaction; The arguments are unique to the transaction type; The chaincode executes the transaction, reads and writes entries in its state as required, and reports whether it was successful or unsuccessful;
- Query transaction: Returns a state entry directly from reading the persistent state of a peer; this may or may not guarantee linearizability.

The fabric includes a security framework for authentication and authorization since it implements a permissioned ledger. Enrollment and transaction authorization are supported by public-key certificates, and chaincode security is ensured by in-band encryption.

3.4.2. Hyperledger Composer

According to (Dahmen & Liermann, 2019), Hyperledger Composer is a software development platform for creating business networks at a high level of abstraction. Using a simple scripting language and a graphical user interface, smart contract logic and applications with interfaces to other resources can be created.

The three main components of Hyperledger Composer are:

- The modeling language which is used to define the participants and the assets
- The transactions which interact with the participants and the assets
- The rights to access data and transactions.

Hyperledger Composer is a highly powerful platform for Proof of Concepts (PoCs) and prototypes that make use of the Hyperledger Fabric API.

3.4.3. Hyperledger Convector

According to (*Getting Started - Covalent Documentation*, n.d.), the Convector Suite is an Open Source Suite for Enterprise Blockchain Networks. It is composed of a group of Development tools for Hyperledger Fabric and its main purpose is to be an agnostic toolset.

Convector Suite main components are:

- Convector Smart Contracts - is a JavaScript-based Development Framework for Enterprise Smart Contract Systems. By abstracting complexities, it aims to make it easier for developers to develop, test, and deploy enterprise-grade smart contract systems.
- Hurley - is the development environment toolset for blockchain projects. It supports Hyperledger Fabric and is being ported to support other chain technologies.

As a rapid development platform for prototypes and proofs of concept, the Convector system can take the place of Hyperledger Composer. Convector also has the advantage of supporting more than just the Hyperledger Fabric platform (Dahmen & Liermann, 2019).

3.5. Tools for Blockchain Benchmarking

In this chapter, some tools for blockchain benchmarking will be presented as well as a few key features.

3.5.1. Hyperledger Caliper

According to (Ampel et al., 2019), Hyperledger Caliper is a tool for measuring the efficiency of permissioned blockchains. It helps users to compare and contrast various blockchains in similar environments.

Caliper is capable of running benchmarks against various blockchain platforms. Caliper was designed for extensibility, allowing it to work with today's most popular monitoring and infrastructure solutions (*Hyperledger Caliper Architecture*, n.d.).

Throughput, latency, performance rate, and CPU / Memory resource utilization are all metrics that the tool can monitor. This is accomplished by listening for transaction timestamps and measuring metrics from them. Throughput is measured in transactions per second and indicates how quickly transactions are effectively committed to the ledger. The interval between sending and receiving transactions is calculated in seconds and is referred to as latency. The success rate is the proportion of transactions that were successfully committed to the total number of transactions sent. The target for this metric should be 100%. The minimum, maximum, and average utilization

of those indicators can be found in CPU / Memory resource consumption. Memory is expressed in megabytes and CPU is calculated as a percentage.(Ampel et al., 2019).

Caliper generates a report based on the SUT responses observed. The following diagram (Fig. 9) depicts this oversimplified viewpoint. (*Hyperledger Caliper Architecture*, n.d.).

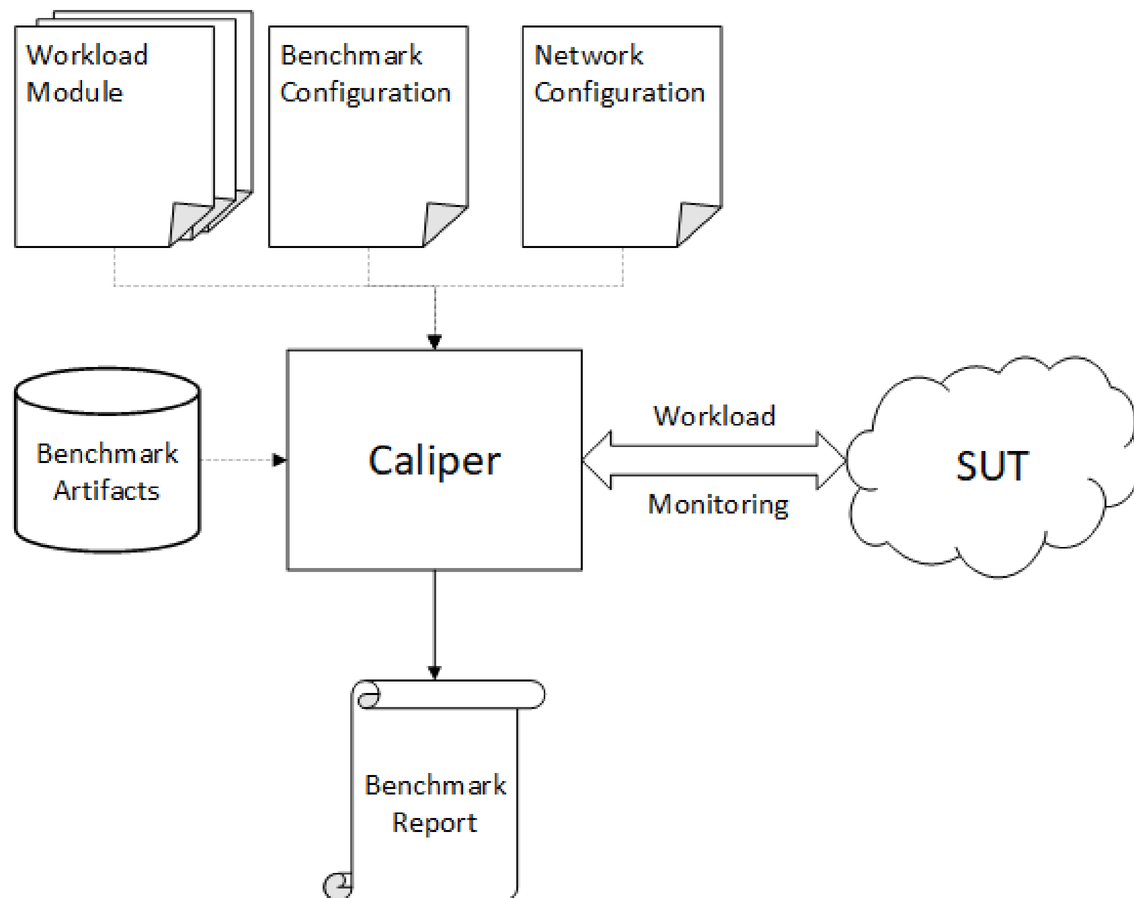


*Figure 9 - Hyperledger Caliper Architecture, retrieved from (*Hyperledger Caliper Architecture*, n.d.)*

3.5.2. Blockbench

According to (Wang et al., 2019), Blockbench is the first benchmark for examining and comparing private blockchain results. Blockbench focuses on the private blockchain because the efficiency of public blockchain has been extensively studied. The consensus layer, data model layer, execution layer, and application layer are the four layers that the author abstracts from the blockchain. Blockbench measures back-end device efficiency in four dimensions: throughput, latency, scalability, and fault tolerance. The following image (Fig. 10) portrays Blockbench's architecture.

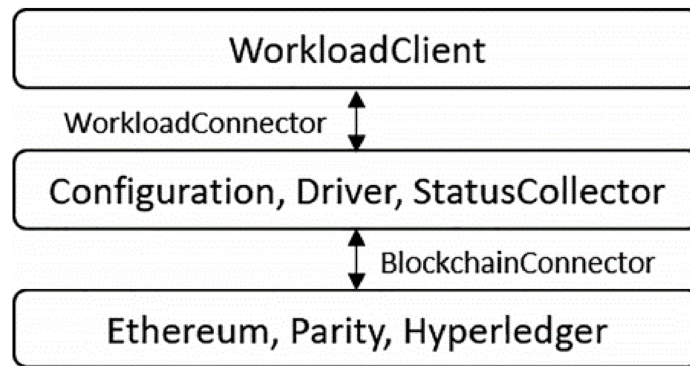


Figure 10 - Architecture of Blockbench (Wang et al., 2019)

Five macro benchmark workloads such as(key-value storage, OLTP (Smallbank), EtherId, Doubler, and WavesPresale and four micro benchmark workloads such as DoNothing, Analytics, IOHeavy, and CPUHeavy have been developed by Blockbench (Wang et al., 2019).

3.5.3. Prometheus

According to (Prometheus, 2019), SoundCloud created Prometheus, an open-source device monitoring and alerting toolkit. Many businesses and organizations have embraced Prometheus since its launch in 2012, and the project has a thriving developer and user community. It is now a self-contained open-source project that is run without the involvement of any organization.

Prometheus has a query language and a basic but powerful data model that allows you to analyze how your applications and infrastructure are performing (Brazil, 2018).

The following are the major characteristics of Prometheus (Prometheus, 2019):

- A multi-dimensional data model with time series data identified by metric name and key/value pairs
- PromQL, a flexible query language to leverage this dimensionality
- No need for distributed storage; single server nodes are self-contained
- Time series data is collected using a pull model over HTTP.
- An intermediate gateway is used to push time series.
- Service discovery or static configuration are used to find targets.
- Support for various graphing and dashboarding modes

Multiple components make up the Prometheus ecosystem, many of which are optional:

- Prometheus' main server, which scrapes and saves time-series data
- Application code instrumentation client libraries
- A push gateway for supporting short-lived jobs
- Exporters specialized in services like HAProxy, StatsD, Graphite, etc.
- An alert manager to handle alerts
- Multiple support tools

For short-lived jobs, Prometheus scrapes metrics from instrumented jobs, either directly or through an intermediary push gateway. It saves all scraped samples locally and applies rules to them in order to aggregate and record new time series from existing data or to generate alerts. The collected data can be visualized using Grafana or other API users. Prometheus is an excellent tool for capturing strictly numerical time series. It can be used for both machine-centric and highly complex service-oriented design monitoring. Its support for multi-dimensional data collection and querying is a specific strength in the world of microservices. Prometheus is designed for reliability. Each Prometheus server is self-contained, meaning it is not reliant on network storage or other third-party services. When other parts of your infrastructure fail, you can depend on it, and you don't need to set up a lot of infrastructures to use it (Prometheus, 2019).

Prometheus is a quick and easy-to-use system. Millions of samples per second can be ingested by a single Prometheus server. It is a single binary that is statically connected and has a configuration file. Prometheus' components can all be run in containers, and they don't do anything fancy that would obstruct configuration management software. It's made to fit with the already existing infrastructure. It is not intended to be a management tool in and of itself, but rather to be incorporated into and developed on top of the existing infrastructure (Brazil, 2018).

The following image (Fig. 11) describes Prometheus' Architecture (Prometheus, 2019):

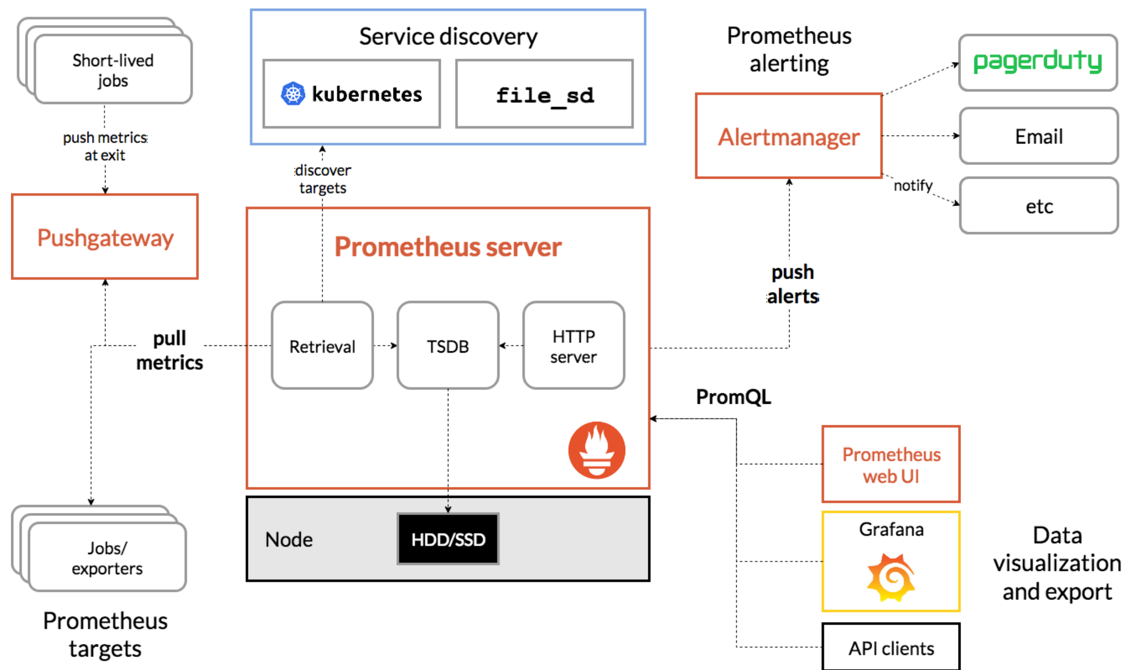


Figure 11 - Prometheus Architecture (Prometheus, 2019)

4. Research Methodologies

This chapter describes the methodology used as a fundamental part of this work. The methodology in question is Design Science Research (DSR).

4.1. Design Science Research

To elaborate on this dissertation, Design Science Research in the domain of Information Systems will be used. According to (Hevner & Chatterjee, 2010) Design Science Research is the systematic paradigm in which a designer uses innovative artifacts to address questions about human challenges, thus adding new insight to the body of scientific evidence. The crafted objects are both helpful and essential to comprehending the issue.

According to (Peffers et al., 2008) the Design Science Research methodology incorporates six activities in a nominal sequence. The activities are the following:

1. **Problem identification and motivation:** This practice entails defining the research problem as well as justifying the solution value. The problem description will be used to create an artifact, which will then be used to provide a viable solution. It can also be useful for breaking down the problem into basic concepts, allowing the solution to capture its complexity. Justifying the worth of a solution not only encourages the researcher and audience to follow it and consider the findings, but it also aids in understanding the logic behind the researcher's understanding of the issue.

In this project, the main problem is that, in Healthcare, there is no secure way of sharing and storing data. Data can be modified and deleted, simply by having administrator permissions. Furthermore, there is no historical log of data currently that satisfies the requirements previously stated. Blockchain technology is based on the immutability and reliability of data and the historical timeline of data storage, therefore it is a viable solution for this problem.

2. **Define the objectives for a solution:** The goals of a solution are rationally inferred from the problem description and awareness of what is practicable and feasible in this operation. The goals can be quantitative or qualitative. Awareness of the current state of problems, as well as current solutions and their effectiveness, are necessary resources.

The main objective is assuring a secure way for data to be shared among agencies as well as data's immutability, reliability, and historical timeline.

3. **Design and development:** The artifact's development is the focus of this activity. A design research artifact is any built object that incorporates a research input into the design. This practice entails deciding the desired functionality and architecture of the artifact. Knowledge of theory that can be applied in a solution is one of the tools needed for moving from goals to design and production.

Regarding this project, firstly, there is the need to research to obtain more knowledge about the process of developing a blockchain-based platform and the tools and frameworks necessary for it.

4. **Demonstration:** The use of the artifact to solve one or more instances of the problem is demonstrated in this operation. This may include practices such as simulation, experimentation, case study, evidence, and others. The successful knowledge of how to use the artifact to solve the problem is one of the necessary tools.

In this project, after the platform is developed, tests and benchmarks will be conducted to assure that the platform satisfies the problems' objectives.

5. **Evaluation:** This practice entails observing and calculating how well the artifact supports a problem solution. It entails matching a solution's goals to actual observed outcomes from the demonstration's use of the artifact. It necessitates an understanding of applicable metrics and measurement methods. In other words, it can be shown if the goals were met in this operation.

In this project, after the platform is developed, there will be a comparison between the objectives previously inferred and the quality of the artifact, to analyze whether the objectives were met.

6. **Communication:** This practice entails informing researchers and other interested parties about the issue and its significance, as well as the artifact and its usefulness, nature, and effectiveness. In a nutshell, this operation entails the exchange of problem and artifact information.

Regarding this project, in the final phase, a presentation will be conducted about the artifact developed. The presentation will include the obtained knowledge, provided by the research itself and the developed artifact.

The following diagram (Fig. 12) shows the six activities that compose this methodology.

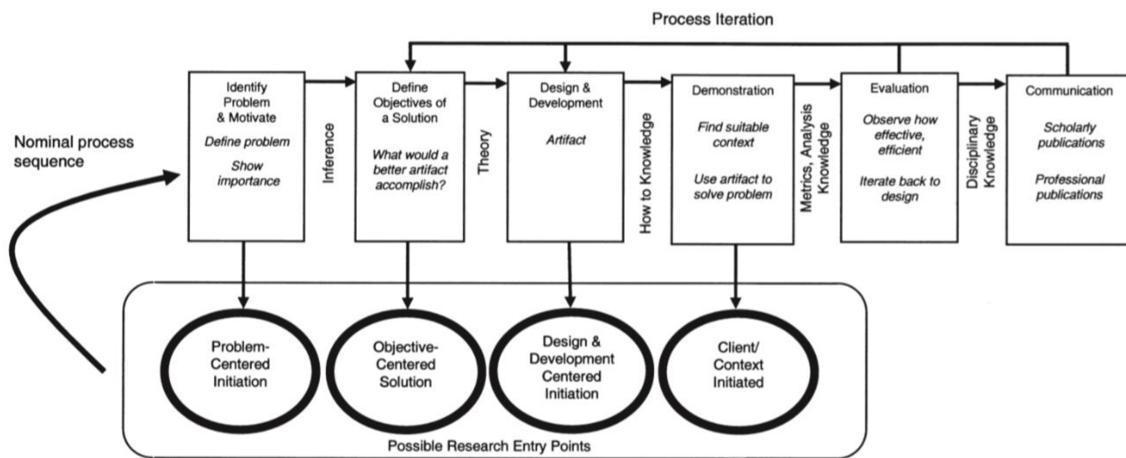


Figure 12 - Design Science Research diagram (Peffers et al., 2008)

According to (Peffers et al., 2008), although the process is structured in sequential order, there is no need to follow a determined order.

5. Project Development

This chapter describes the project development process as well as all the technologies and tools used to create a functional prototype.

This project consists of the development of a private blockchain structure to support medical data gathering. Patients have beacons in their rooms that interact with an app that runs on medical staff's tablets. This app communicates with the blockchain through a REST API that provides various functions for asset creation in the blockchain.

5.1. Tools and Frameworks used

This chapter comprehends all the Tools and Frameworks used for the development of the project as well as a brief description of the context of usage of said Tools and Frameworks.

5.1.1. Hypeledger Fabric

Hyperledger Fabric is designed to serve as a platform for building modular applications and solutions. Plug-and-play modules, such as consensus and membership services, are possible with Hyperledger Fabric. Its modular and adaptable architecture caters to a wide variety of industry applications. It takes a novel approach to consensus that allows for scalability while maintaining privacy (The Linux Foundation, 2020c).

Hyperledger Fabric was chosen as the blockchain framework for this project as it is an open-source project which offers all the necessary tools to deploy a private blockchain.

5.1.2. Hyperledger Caliper

Hyperledger Caliper is a blockchain benchmarking tool that allows users to assess a blockchain implementation's success against a collection of predefined use cases. Hyperledger Caliper can generate reports with a variety of performance metrics (The Linux Foundation, 2020a).

Hyperledger Caliper was the chosen benchmarking tool for blockchain since it offers powerful metrics, it's easy to set up, and is developed by Hyperledger, which makes integration with Hyperledger Fabric easy and seamless.

5.1.3. Prometheus and Grafana

Prometheus is an open-source system monitoring and alerting toolkit (Prometheus, 2019).

In this project, Prometheus was used to benchmark the computational load such as the CPU and Memory usage of every component of the blockchain. Prometheus retrieved metrics for every docker container while Grafana provided Graphs and Dashboards for these metrics.

5.1.4. Blockchain Explorer

Blockchain Explorer is a user-friendly Web application platform for viewing, invoking, deploying, and querying blocks, transactions, and related data, network information (name, status, list of nodes), chain codes and transaction families, and any other applicable data stored in the ledger (The Linux Foundation, 2020b).

In this project, it was used to better visualize the blockchain. In the app, you can query blocks and transactions as well as see the network peers and orderers, the chaincodes installed and their respective versions, and the channels available.

5.1.5. Visual Studio Code

Visual Studio Code is a reimagined and streamlined code editor for developing and debugging modern web and cloud applications (Microsoft, 2020).

In this project, it was used to edit Hyperledger Fabric files such as javascript files, config files, yaml files, go files, etc. Moreover, it served to better visualize the folder and project structure, keeping the developing environment more organized and therefore, easier to navigate. This editor was chosen for its simplicity, lightweight, and modern interface.

5.1.6. Go Language

Go is an open-source programming language that makes it simple to create software that is reliable and powerful (Meyerson, 2014).

The Go language was used to create the smart contracts that integrate Hyperledger Fabric. All the functions of asset query and asset creation were coded in go as well as the asset structs.

5.1.7. NodeJS

Node.js is a scalable network application builder that uses an asynchronous event-driven JavaScript runtime. Many connections can be managed at the same time (OpenJS Foundation, 2020).

NodeJS was chosen as a language for the development of the Rest API because it offers fast development, easy deployment, and consistent network capabilities.

5.1.8. Postman

According to (Postman Inc., 2021), Postman is an API development collaboration tool. Postman's features make each stage of creating an API easier to understand and collaborate on, allowing you to develop better APIs faster.

In this project, Postman was used to better organize the API functions as well as facilitate the development of the API. It creates a savable environment in which functions, URLs, request bodies, params, and authentication tokens are saved. With this in mind, Postman was used to creating a collection specific to the project API.

5.2. Prerequisites

This chapter contains all the software required to develop this project as well as the instructions required to install it correctly. In some cases, to run Hyperledger Fabric specific versions of the software are required. Some ways of verifying if every tool is installed correctly are also provided in this chapter.

To develop this project the following tools were installed:

- Operating System: Linux Ubuntu 20.04 LTS 64-bit
- cURL tool: Latest version (used version 7.68.0)
- git (used version 2.25.1)
- Docker engine: latest version (used version Docker 20.10.5, build 55c4c88)
- Docker Compose: lastest version (used version docker-compose 1.25.0)
- Node: latest version (used version v10.19.0)
- npm: Version 8.9 or higher, version 9 to 10.15.2 are not supported (used version 6.14.4)
- Python: version 2.7.x (used version Python 2.7.18)
- Go: Version 1.13.x (used version go 1.13.8)
- Vim (used version 8.1.2269)
- Visual Studio Code (used version 1.54.3)
- Postman (used version 7.36.5)

Firstly, curl and golang software package were installed. For this, the following commands were executed in the terminal:

- sudo apt-get install curl
- sudo apt-get install golang

- export GOPATH=$HOME/go
- export PATH=$PATH:$GOPATH/bin
- sudo apt-get install vim
- sudo snap install postman
- sudo snap install –classic code

To be able to call golang in a terminal anywhere in the operating system, the path must be added to the bashrc file. The path of the local golang installation was provided by editing the bashrc file with a text editor of choice. In this case, "vim" was used as the text editor.

The following command was used to edit the bashrc:

- vim ~/.bashrc

The following lines of code were added to the bashrc file:

- export PATH=$PATH:/usr/local/go/bin:/home/hugo/.go/bin
- export GOPATH=/home/hugo/go
- export GOROOT=/usr/local/go
- export PATH=$PATH:$GOPATH/bin

After editing and saving the bashrc file, the following command was executed in order to update the environment so that the changes made in the bashrc file are reflected on the current shell:

- source ~/.bashrc

This prevents having to reopen the shell.

After sourcing the bashrc file, NodeJS, npm and Python were installed.

To accomplish this, the following commands were executed:

- sudo apt-get install nodejs
- sudo apt-get install npm
- sudo apt-get install python

After installing the previously mentioned software, golang was installed manually to use the 1.13.x version. The go compressed files were downloaded, extracted and moved to the /usr/local folder. To perform this task the following commands were executed by sequential order:

- wget https://dl.google.com/go/go1.13.x.linux-amd64.tar.gz\

- tar -xzvf go1.13.x.linux-amd64.tar.gz
- sudo mv go/ /usr/local
- export GOPATH=/usr/local/go
- export PATH=$PATH:$GOPATH/bin

After successfully installing the previously mentioned software, both docker and docker-compose were installed. To perform this task, the following commands were executed by sequential order in the terminal:

- curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add –
- sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu
- $(lsb_release -cs) stable"
- sudo apt-get update
- apt-cache policy docker-ce
- apt-get install -y docker-ce
- sudo apt-get install docker-compose
- sudo apt-get upgrade

After docker is installed successfully, to enable the docker service on boot the following command was executed:

- sudo systemctl enable docker.service

Finally, to start the docker service the following command was executed:

- sudo systemctl start docker

After having installed docker successfully, the following commands were executed to verify that the software versions installed are the correct ones:

- curl -V
- npm -v
- docker version
- docker-compose version
- go version

– python -V
– node -v

After verifying that all the previous software was installed correctly and that the versions installed were correct, samples, binaries, and docker images were installed. These binaries have different purposes that are crucial for the successful development of the project. The following list comprehends the various binaries and their respective purposes:

– Configtxgen – Creating network artifacts(genesis.block/channel.tx)
– Configtxlator – Utility for generating channel configuration
– Cryptogen – Utility for creating key material
– Discovery – Command-line client for service discovery
– Idemixgen – Utility for generating key material to be used with identity mixer MSP
– Orderer – node
– Peer – node
– Fabric-ca-client – Client for creating, registering, and enrolling users

The following commands were used to create a folder and download the samples, binaries, and docker images:

– mkdir testfolder
– cd testfolder
– curl -sSL https://bit.ly/2ysbOFE | bash -s
– curl sSL https://bit.ly/2ysbOFE | bash -s - fabric_version fabric-ca_version thirdparty_version
– curl sSL https://bit.ly/2ysbOFE | bash -s - 2.0.1 1.4.6 0.4.18

A *fabric-samples* folder was downloaded in the process. In this folder, there's a subfolder called bin. This folder contains all the fabric binaries necessary for this project. These binaries must be accessible systemwide. For that, the bin folder path was added to the bashrc file. To edit the bashrc the following command was used:

– vim ~/.bashrc

The following line of code was added to the bashrc:

– export PATH=$PATH:/home/hugo/testfolder/fabric-samples/bin

Once again the source command was used to update the environment, so the changes made are reflected on the current shell:

– source ~/.bashrc

To test if everything was set up correctly the command "peer" was executed in the terminal. If everything is set up correctly, the peer command is recognized by the operating system and the following output (Fig. 13) is obtained:

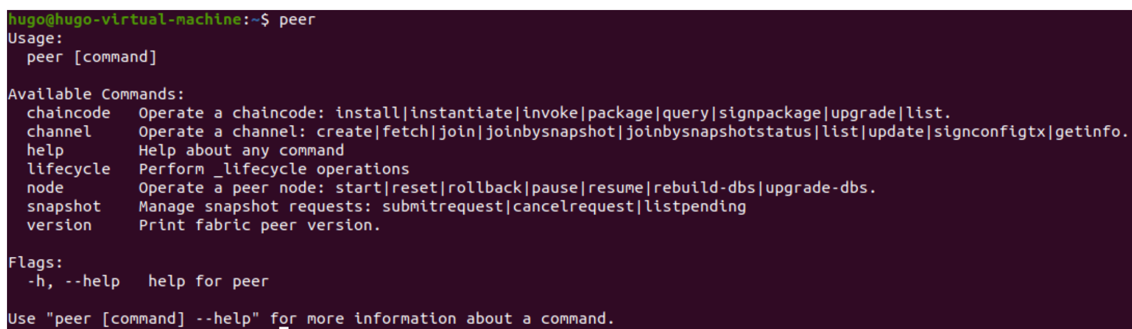


Figure 13 - Peer command execution in order to validate the installation

6. Results and Discussion

This chapter comprehends all the results obtained throughout the project as well as all the details that embody the development of the solution.

6.1. Network Structure

The network proposed was created with the following elements:

– 2 Organizations (Org1 and Org2)
– 2 Peers per Organization (Peer1.Org1, Peer2.Org1, Peer1.Org2, Peer1.Org)
– 3 Orderers (Orderer 1, Orderer 2 and Orderer 3)
– 1 Channel (mychannel)

– 1 Chaincode (tracking)

The following image (Fig. 14) provides a better understanding of the blockchain network architecture created for this project.

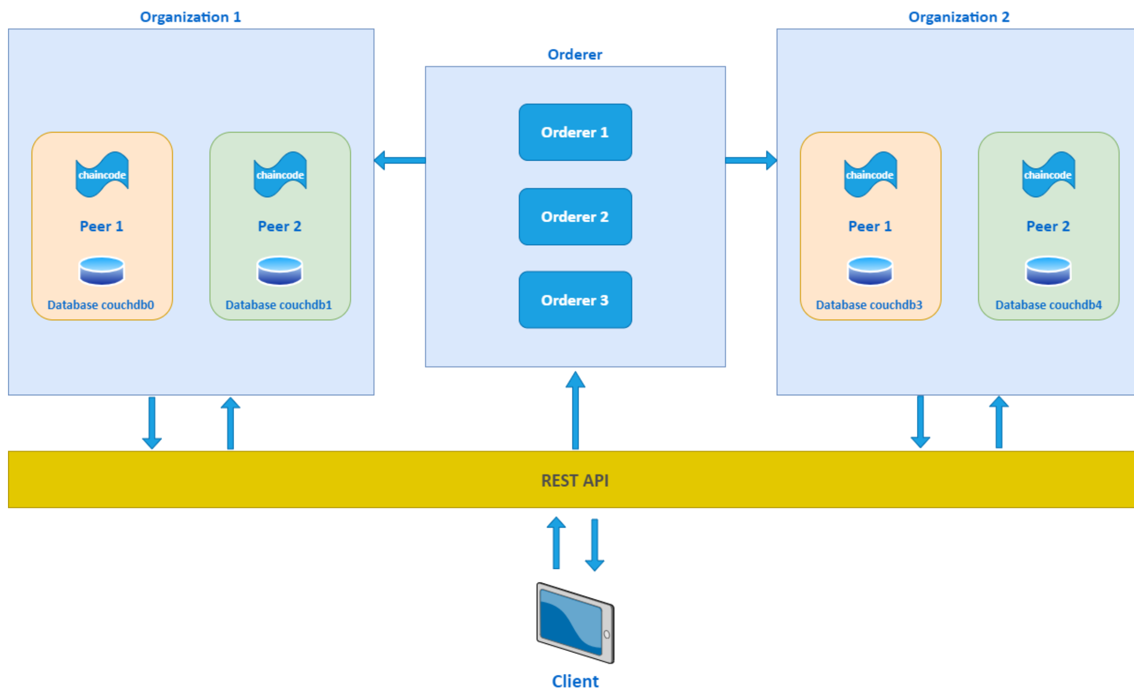


Figure 14 - Blockchain Network Architecture

By observing the previous image (Fig. 14) it can be inferred that the REST API handles the communication between the Client and the blockchain network.

6.2.Hyperledger Fabric

To develop the Hyperledger Fabric part of the project a GitHub repository was downloaded using the following command (Pavan, n.d.-a):

– git clone https://github.com/adhavpavan/BasicNetwork-2.0.git

After cloning the repository, in the folder *"BasicNetwork-2.0/artifacts/channel"* a script named *create-artifacts.sh* was executed in order to genereate the crypto-config which contains all

the certificates and private keys necessary for the blockchain network. To do this task the following command was used:

- ./create-artifacts.sh

The *crypto-config.yaml* file contains the configurations relative to the crypto-config genesis. Depending on the confugration used in this file, private keys and certificiates will be generated accordingly. For this project the configuration for the crypto-config genesis used contains three orderers and two organizations with two peers each.

The file *configtx.yaml* contains configurations such as the anchor peer definition for each organization.

The *docker-compose-yaml* file contains the services definition. This is where all the peers, organizations, oderers and databases are defined and their respective external IP's are defined and mapped to the internal docker container IP.

To start the network the following command was used inside *"BasicNetwork-2.0/artifacts/"* folder:

- docker-compose up -d

To verify that every container defined in the *docker-compose.yaml* is running, the following command was used:

- docker ps

The next step consists in creating a channel and joining all the peers to it. For this task, there is a script called *"createChannel.sh"*. The path for the crypto-config peer mspconfig, peer tls rootcert file, orderer CA, and organizations CA must be provided as well as the channel name and the path of the fabric config file. In this project, this file is located in "BasicNetwork-2.0/artifacts/channel/config ". The channel name used in this project is *mychannel.*

In order to accomplish this task, the previously mentioned script must be executed three times. Firstly, comment the *joinChannel* and *updateAnchorPeers* functions so that the script only executes the *createChannel* function. Execute the script using the following command:

- ./createChannel.sh

After executing the script, comment the *createChannel* function and uncomment the *joinChannel* function and execute the script. Finally, after executing the script for the second time, comment the *joinChannel* function and uncomment the *updateAnchorPeers* function and execute the script for the last time.

To verify that the channel was created successfully and that the peers joined the created channel, we can go inside the docker container of a certain peer and check its channel list. To perform such a task the following commands were used:

- docker exec -it peer0.org1.example.com sh
- peer channel list

The first command is used in order to join the docker container of the peer. Note that *peer0.org1.example.com* is the name of the container used in this project for Peer 0 of Organization 1. The second command should output the name of the previously created channel if everything was set up correctly.

The next step consists in deploying the chaincode. To perform this task, the following script was executed:

- ./deployChaincode.sh

This script follows the fabric 2.0 Chaincode Lifecycle. This Lifecycle is represented in Fig. 15.

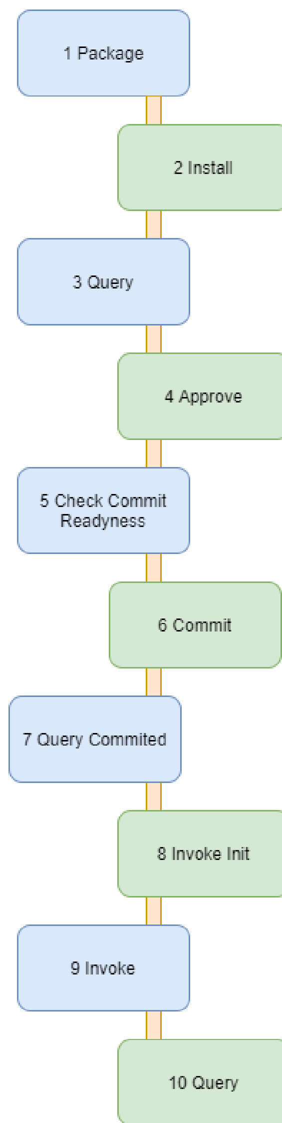


Figure 15 - Chaincode Lifecycle

The script will execute the following functions sequentially, respecting the Chaincode Lifecycle:

- packageChaincode
- installChaincode
- queryInstalled
- approveForMyOrg1
- checkCommitReadyness
- approveForMyOrg2
- checkCommitReadyness

- commitChaincodeDefination
- queryCommitted
- chaincodeInvokeInit
- chaincodeInvoke
- chaincodeQuery

In between some of the functions, there are sleep functions so the system waits some time before executing the next function. This is needed because instantiating the chaincode might take some time and having a delay prevents the next function execution before the first function finishes its execution.

The same paths used to create the channel are used to deploy the chaincode, so the environmental variables used in this step are the same. The name and path to the smart contract must be specified in the script.

The following list contains the definitions used:

- CHANNEL_NAME="mychannel"
- CC_RUNTIME_LANGUAGE="golang"
- VERSION="1"
- CC_SRC_PATH="./artifacts/src/github.com/tracking/go"
- CC_NAME="tracking"

To create the Smart Contract, the *fabcar* example was copied and modified to accommodate the tracking structure.

The Smart Contract follows the structure represented in the following table diagram (Table 2).

Table 2 - API structure table diagram

Beacon	Dispositivo Médico (Medical Device)	Médico (Doctor)	Doente (Patient)
ID Beacon	ID Dispositivo (ID Device)	ID Médico (ID Doctor)	ID Doente (ID Patient)
V: Sala (Room)	Nome (Name)	Nome (Name)	V: Sala (Room)
	V: ID Médico (ID Doctor)		
	V: ID Doente (ID Patient)		
	V: Sala (Room)		

The *V* prefix stands for Variable and marks the properties that need to have the ability to be changed which translates into a new state for the asset. Both the beacon and patient can be physically moved. For instance, the patient can be moved to another room for a different treatment and so does the beacon. The Medical Device can be associated with a different doctor in case of multiple doctors using the same device. The room can also change and so does the patient being tracked.

For the implementation, a private Hyperledger Fabric blockchain was employed in a Linux Ubuntu virtual machine. A smart contract was created to define the data structures and functions necessary for the prototype. For instance, a struct named *DispositivoMedico* was created in the smart contract in order to define the medical device data structure that will be stored in the tracking process. In the same line of thought, a function named *createDispMedico* was developed for the creation of the medical device asset as well as its insertion in the blockchain. Various functions for asset creation and asset state change were developed for this prototype. As presented in Table 3, each function as well as its name and purpose was specified.

Table 3 - Functions for asset creation and asset state change

Name	Function	Purpose
Create Beacon	createBeacon	Create a Beacon
Create Doctor	createMedico	Create a Doctor
Create Medical Device	createDispMedico	Create a Medical Device
Create Patient	createDoente	Create a Patient
Change Beacon Room	changeBeaconSala	Change the Room of a Beacon
Change Patient Room	changeDoenteSala	Change the Room of a Pacient
Change Medical Device Doctor	changeDispMedMedico	Change the Doctor of a Medical Device
Change Medical Device Room	changeDispMedSala	Change the Room of a Medical Device
Change Medical Device Patient	changeDispMedDoente	Change the Patient of a Medical Device
Get History Of Asset	getHistoryForAsset	Get the state history of a certain asset

This type of structure allows asset creation and asset state updates as well as viewing the asset state history which helps in the tracking process. For instance, you can check which rooms a given patient has been in, or which doctors had access to a certain medical device and which patient they were treating at a certain time.

Fig 16 represents the structs created in the Smart Contract:

```
17  // SmartContract Define the Smart Contract structure
18  type SmartContract struct {
19  }
20
21  type Beacon struct {
22      Sala string `json:"sala"`
23  }
24
25  type DispositivoMedico struct {
26      Nome   string `json:"nome"`
27      Medico string `json:"medico"`
28      Sala   string `json:"sala"`
29      Doente string `json:"doente"`
30  }
31
32  type Medico struct{
33      Nome string `json:"nome"`
34  }
35
36   type Doente struct{
37      Sala string `json:"sala"`
38   }
```

Figure 16 - Smart Contract structs

Fig 17 represents the Smart Contract function that creates the Dispositivo Medico asset:

```
117  func (s *SmartContract) createDispMedico(APIstub shim.ChaincodeStubInterface, args []string) sc.Response {
118
119      if len(args) != 5 {
120          return shim.Error("Incorrect number of arguments. Expecting 5")
121      }
122
123      var dispMed = DispositivoMedico{Nome: args[1], Medico: args[2], Sala: args[3], Doente: args[4]}
124
125      dispMedAsBytes, _ := json.Marshal(dispMed)
126      APIstub.PutState(args[0], dispMedAsBytes)
127
128
129      return shim.Success(dispMedAsBytes)
130  }
```

Figure 17 - Smart Contract function createDispMedico

Fig 18 represents the Smart Contract function that changes the Doente of the Dispositivo Medico.

```
216 func (s *SmartContract) changeDispMedDoente(APIstub shim.ChaincodeStubInterface, args []string) sc.Response {
217
218     if len(args) != 2 {
219         return shim.Error("Incorrect number of arguments. Expecting 2")
220     }
221
222     dispMedAsBytes, _ := APIstub.GetState(args[0])
223     dispMed := DispositivoMedico{}
224
225     json.Unmarshal(dispMedAsBytes, &dispMed)
226     dispMed.Doente = args[1]
227
228     dispMedAsBytes, _ = json.Marshal(dispMed)
229     APIstub.PutState(args[0], dispMedAsBytes)
230
231     return shim.Success(dispMedAsBytes)
232 }
```

Figure 18 - Smart Contract function changeDispMedDoente

Fig 19 represents the Smart Contract function that queries the blockchain and returns the state history of an asset:

```
252 func (t *SmartContract) getHistoryForAsset(stub shim.ChaincodeStubInterface, args []string) sc.Response {
253
254     if len(args) < 1 {
255         return shim.Error("Incorrect number of arguments. Expecting 1")
256     }
257
258     carName := args[0]
259
260     resultsIterator, err := stub.GetHistoryForKey(carName)
261     if err != nil {
262         return shim.Error(err.Error())
263     }
264     defer resultsIterator.Close()
265
266     // buffer is a JSON array containing historic values for the marble
267     var buffer bytes.Buffer
268     buffer.WriteString("[")
269
270     bArrayMemberAlreadyWritten := false
271     for resultsIterator.HasNext() {
272         response, err := resultsIterator.Next()
273         if err != nil {
274             return shim.Error(err.Error())
275         }
276         // Add a comma before array members, suppress it for the first array member
277         if bArrayMemberAlreadyWritten == true {
278             buffer.WriteString(",")
279         }
280         buffer.WriteString("{\"TxId\":")
281         buffer.WriteString("\"")
282         buffer.WriteString(response.TxId)
283         buffer.WriteString("\"")
284
285         buffer.WriteString(", \"Value\":")
286         // if it was a delete operation on given key, then we need to set the
287         //corresponding value null. Else, we will write the response.Value
288         //as-is (as the Value itself a JSON marble)
289         if response.IsDelete {
290             buffer.WriteString("null")
291         } else {
292             buffer.WriteString(string(response.Value))
293         }
294
295         buffer.WriteString(", \"Timestamp\":")
296         buffer.WriteString("\"")
297         buffer.WriteString(time.Unix(response.Timestamp.Seconds, int64(response.Timestamp.Nanos)).String())
298         buffer.WriteString("\"")
299
300         buffer.WriteString(", \"IsDelete\":")
301         buffer.WriteString("\"")
302         buffer.WriteString(strconv.FormatBool(response.IsDelete))
303         buffer.WriteString("\"")
304
305         buffer.WriteString("}")
306         bArrayMemberAlreadyWritten = true
307     }
308     buffer.WriteString("]")
309
310     fmt.Printf("- getHistoryForAsset returning:\n%s\n", buffer.String())
311
312     return shim.Success(buffer.Bytes())
313 }
```

Figure 19 - Smart Contract function getHistoryForAsset

Every function created in the Smart Contract must be added to the Smart Contract Invoke method. The following image (Fig. 20) represents this method:

```
48  // Invoke :  Method for INVOKING smart contract
49  func (s *SmartContract) Invoke(APIstub shim.ChaincodeStubInterface) sc.Response {
50
51      function, args := APIstub.GetFunctionAndParameters()
52      logger.Infof("Function name is:  %d", function)
53      logger.Infof("Args length is : %d", len(args))
54
55      switch function {
56      case "initLedger":
57          return s.initLedger(APIstub)
58      case "getHistoryForAsset":
59          return s.getHistoryForAsset(APIstub, args)
60      case "createBeacon":
61          return s.createBeacon(APIstub, args)
62      case "createDispMedico":
63          return s.createDispMedico(APIstub, args)
64      case "createMedico":
65          return s.createMedico(APIstub, args)
66      case "createDoente":
67          return s.createDoente(APIstub, args)
68      case "changeBeaconSala":
69          return s.changeBeaconSala(APIstub, args)
70      case "changeDispMedMedico":
71          return s.changeDispMedMedico(APIstub, args)
72      case "changeDispMedSala":
73          return s.changeDispMedSala(APIstub, args)
74      case "changeDispMedDoente":
75          return s.changeDispMedDoente(APIstub, args)
76      case "changeDoenteSala":
77          return s.changeDoenteSala(APIstub, args)
78      default:
79          return shim.Error("Invalid Smart Contract function name.")
80      }
81
82  }
```

Figure 20 - Smart Contract Invoke method

Throughout the development phase of this project, there was a need to constantly modify and update the chaincode. To perform this task the following script was executed:

- ./upgradeChaincode.sh

This script is responsible for upgrading and installing the new chaincode. It is very similar to the script used for chaincode deployment as it uses the same definitions and variables. In order to update the chaincode, increment the version number and execute the script. Note that it is important to provide a different version number from the ones already installed in the network, otherwise the execution will fail with an error.

6.3. REST API

This chapter contains the developed API architecture, all the functions created, and their respective behavior. Some images that represent how each function works will be provided as well as database images that confirm that assets were created or changed.

6.3.1. API Architecture

In the folder "*BasicNetwork-2.0/api-1.4/artifacts*" there is a file called *network-config.yaml*. This file contains the network configurations that follow the network structure of the project. The path of the crypto-config private keys and certificates of the Organizations, Peers, Orderers, and Certificate Authorities are specified in this file.

A postman collection was created for this API. This collection contains all the functions developed for the API which directly correlate to the previously created functions in the smart contract. The following table (Table 4) illustrates the postman collection created, mentioning each function created and its request type.

Table 4 - API requests

Request Type	Function
POST	Register User
POST	Create Beacon
POST	Create Dispositivo Medico
POST	Create Medico
POST	Create Doente
POST	Change Beacon Sala
POST	Change Dispositivo Medico Medico
POST	Change Dispositivo Medico Sala
POST	Change Dispositivo Medico Doente
POST	Change Doente Sala
GET	Get History Of Asset

As it can be inferred from Table 3, eleven functions were created, one of which is a GET request and the rest are POST requests.

The REST API was made with a generical approach so that the Smart Contract logic can be changed without the need to change the API. This results in the need to provide the API with

certain parameters on each API call. The parameters are sent in the request body in jSON format and have the following structure:

- ```
  {
      "fcn": "function_name",
      "peers": ["peer_name_1", "peer_name_2", ...],
      "chaincodeName": "chaincode_name",
      "channelName": "channel_name",
      "args": ["arg0", "arg1", ...]
  }
  ```

The URL for the requests has the following structure:

- http://localhost:400/channels/{channel_name}/chaincodes/{chaincode_name}

The *{channel_name}* and the *{chaincode_name}* must be replaced to the correspondent project channel and chaincode in use. For this project the channel used is called *mychannel* and the chaincode used is called *tracking*.

This structure allows the API to support various channels and chaincodes independently.

Before running the API, the node modules must be installed first. To accomplish this task in the folder "BasicNetwork-2.0/api-1.4" the following command was run:

- npm install

After the node modules were installed successfully, the API was started using the following command:

- node app.js

For this project, the API runs in the http://localhost:4000 domain.

6.3.2. Register User and Authentication Token

Every API call needs to be authenticated with a bearer token. This token is obtained upon user registration. To register a user, both username and organization name must be provided in the request body. The URL used for the request is http://localhost:4000/users.

The following image (Fig. 21) shows the process of registering a user:
  ```

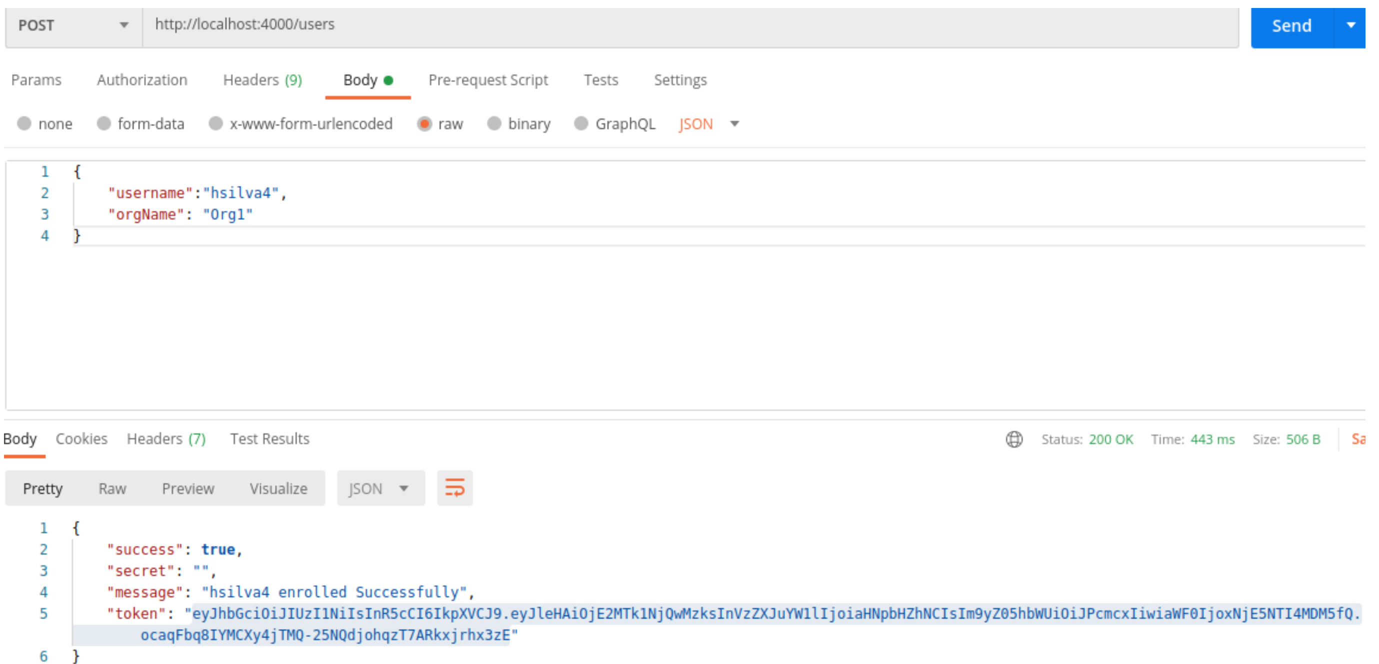


Figure 21 - Create a User

As can be inferred from the figure above, the response body returns the API request success status and the Bearer Token that will be used for all future API calls.

As all API calls need an authentication token, the token must be provided for every function. The following image (Fig. 22) shows how to accomplish such a task.

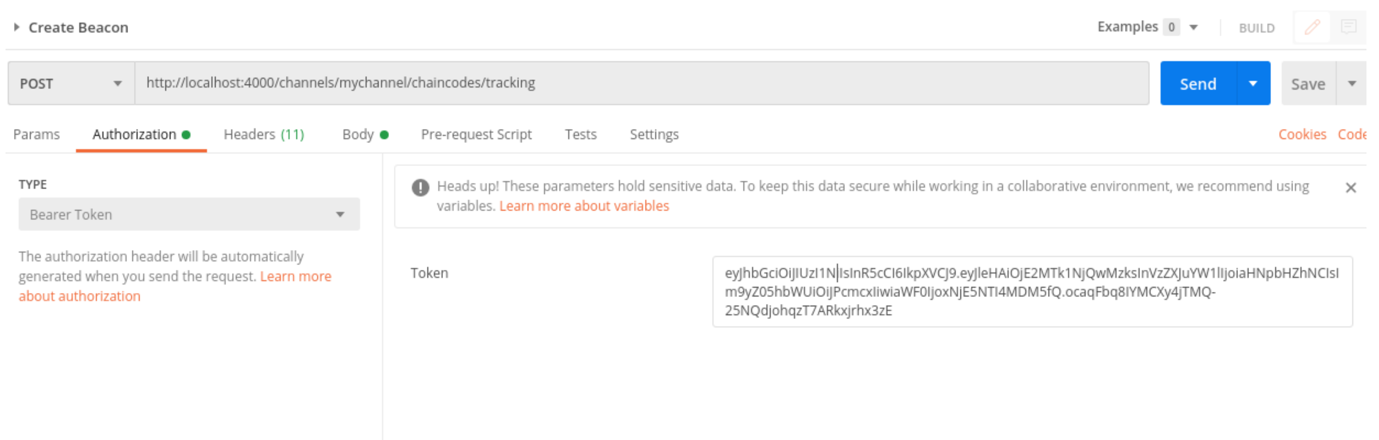


Figure 22 - Bearer Token being defined in the Create Beacon function

6.3.3. Create Beacon (createBeacon)

The function *createBeacon* was developed to register a beacon in the blockchain network, as the project raises a necessity to associate a beacon with a room. This function takes the following arguments for the creation of the beacon asset:

- Arg1: Beacon ID (id)
- Arg2: Room (sala)

The following image (Fig. 23) shows a POST request made to the API with the function *createBeacon*.

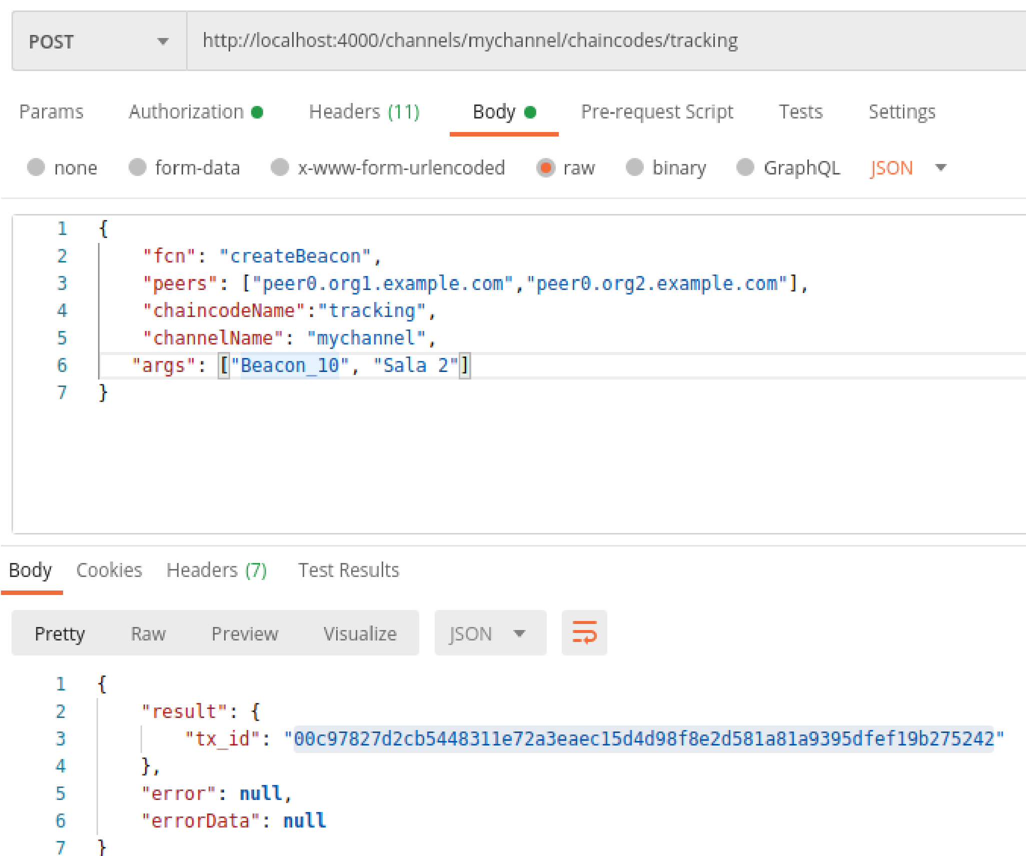


Figure 23 - createBeacon function API call

As it can be inferred from the image above, if the POST request is successful, a Transaction ID will be received in the response body under the name *tx_id*. If the request is unsuccessful an error message is displayed. This type of behavior was implemented in all the functions created.

6.3.4. Create Doctor (createMedico)

The function *createMedico* was developed to register a Doctor in the blockchain. This function takes the following arguments:

- Arg1: Doctor ID (id)
- Arg2: Doctor Name (nome)

The following image (Fig. 24) shows a POST request made to the API with the function *createMedico*:

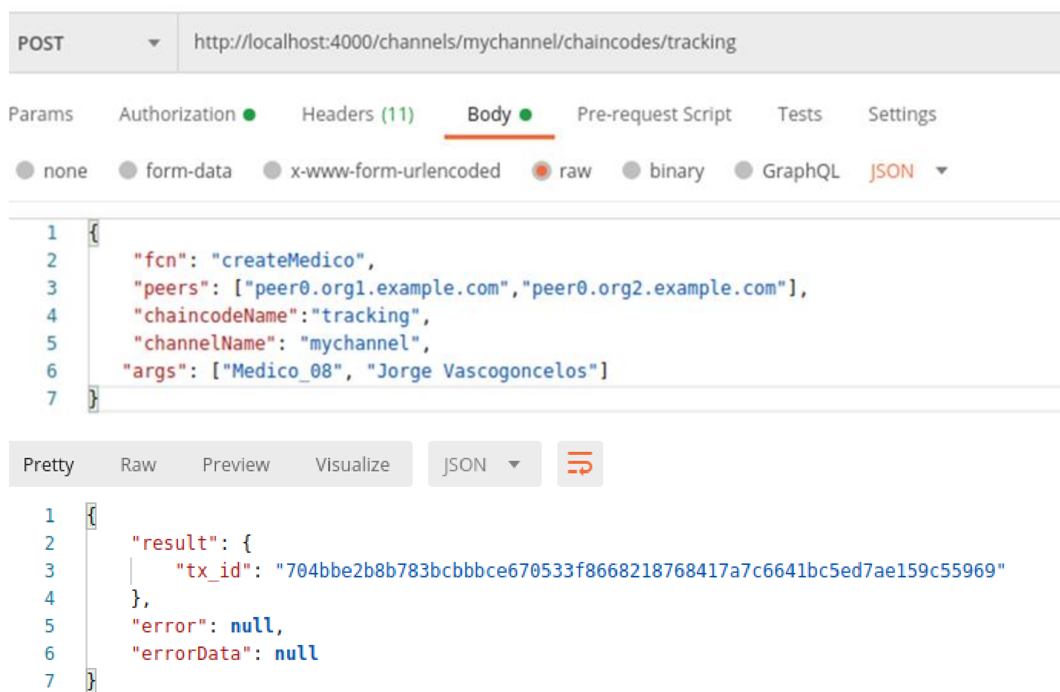


Figure 24 - createMedico function API call

The following image (Fig. 25) represents the recently created asset in the database.

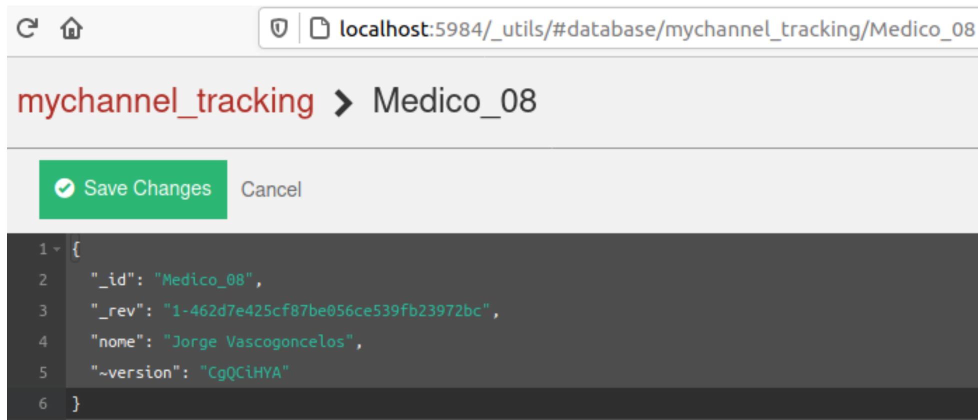


Figure 25 - Medico_08 asset in the database

6.3.5. Create Patient (createDoente)

The function *createDoente* was developed to register a patient in the blockchain. The project raised a necessity to associate patients with a certain room. This function takes the following arguments:

- Arg1: Patient ID (id)
- Arg2: Room (sala)

The following image (Fig. 26) shows a POST request made to the API with the function *createDoente*:

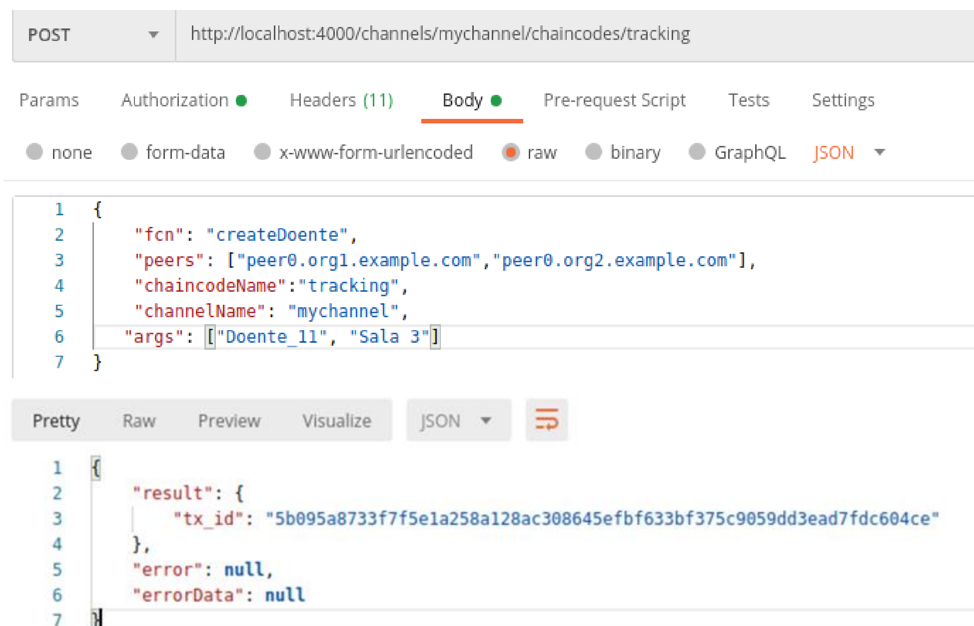


Figure 26 - createDoente function API call

6.3.6. Create Medical Device (createDispMedico)

As the project raised a necessity to register medical devices in the ledger, the function *createDispMedico* was developed. This function takes five arguments:

- Dispositive ID (id)
- Dispositive Name (nome)
- Doctor (medico)
- Room (sala)
- Patient (doente)

The following image (Fig. 27) shows a POST request made to the API with the function *createDispMedico*:

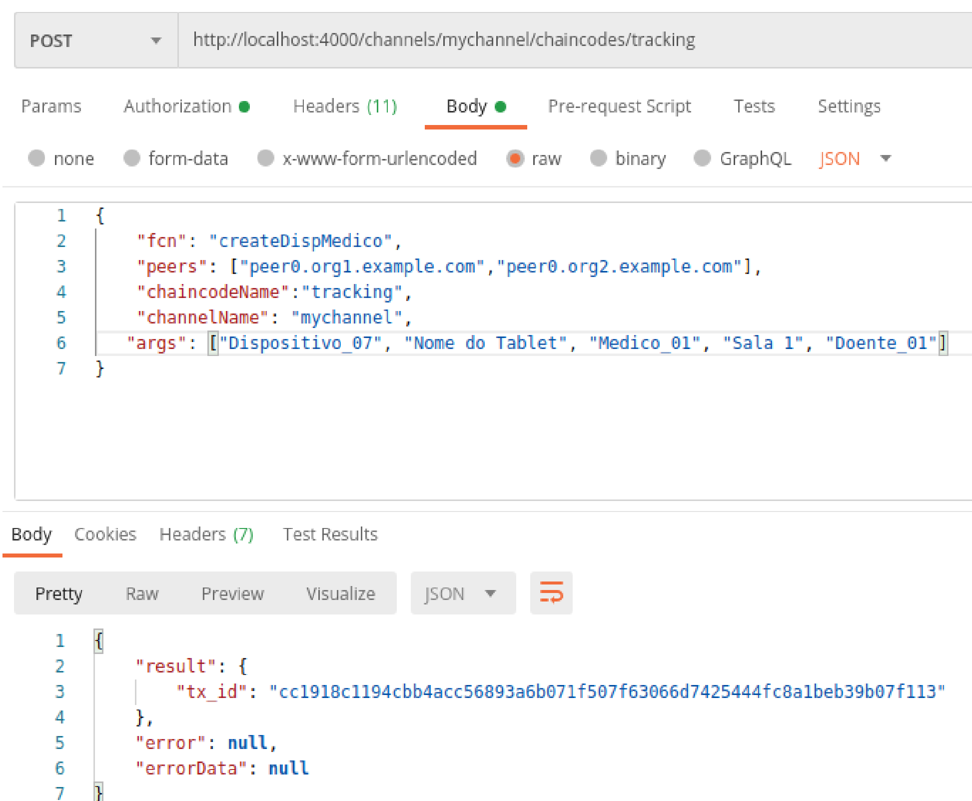


Figure 27 - createDispMedico function API call

6.3.7. Change Beacon room property (changeBeaconSala)

As previously stated some asset arguments present a need to be changed.

The *changeBeaconSala* function allows changing the room (sala) property of any stored Beacon asset in the blockchain network. This function takes the two following arguments:

- Arg1: Beacon ID (id)
- Arg2: Room (sala)

The first argument is used to identify the asset and the second argument is the property that will be changed.

The following image (Fig. 28) shows a POST request made to the API with the function *changeBeaconSala*:

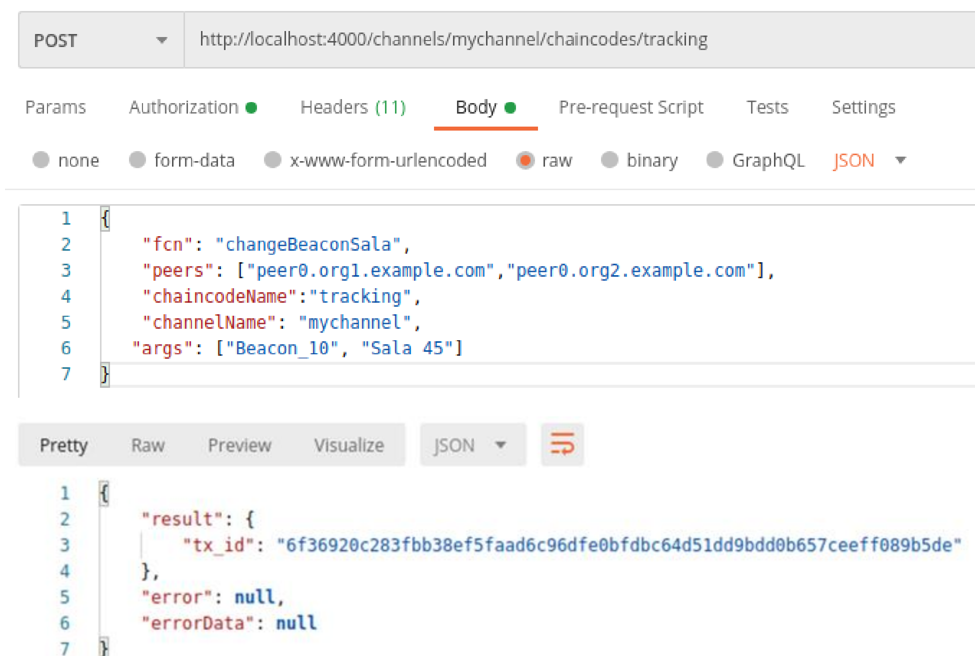


Figure 28 - changeBeaconSala function API call

The following image (Fig .29) represents the recently changed asset in the database:

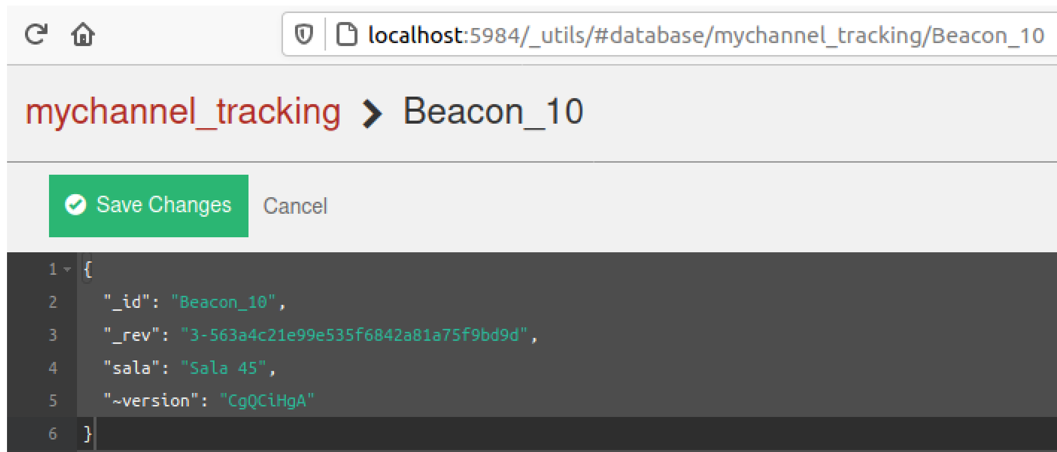


Figure 29 - Beacon_10 asset in the database

6.3.8. Change Patient room property (changeDoenteSala)

The *changeDoenteSala* function allows changing the room (sala) property of any stored Patient (Doente) asset in the blockchain network. This function takes the two following arguments:

- Arg1: Patient ID (id)
- Arg2: Room (sala)

The first argument is used to identify the asset and the second argument is the property that will be changed.

The following image (Fig. 30) shows a POST request made to the API with the function *changeDoenteSala*:

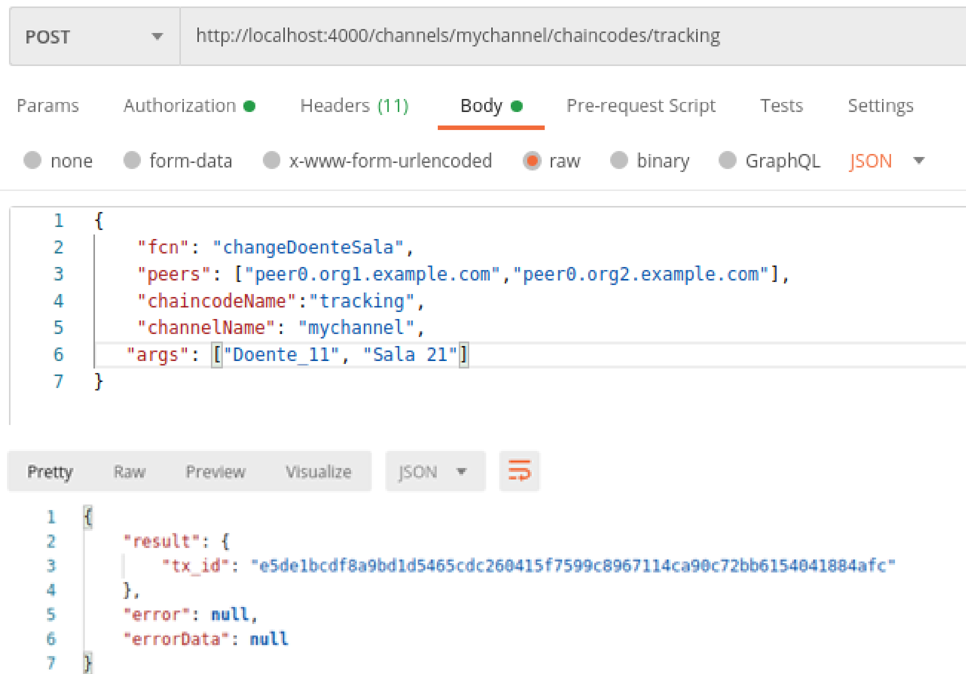


Figure 30 - changeDoenteSala function API call

The following image (Fig. 31) represents the recently changed asset in the database.

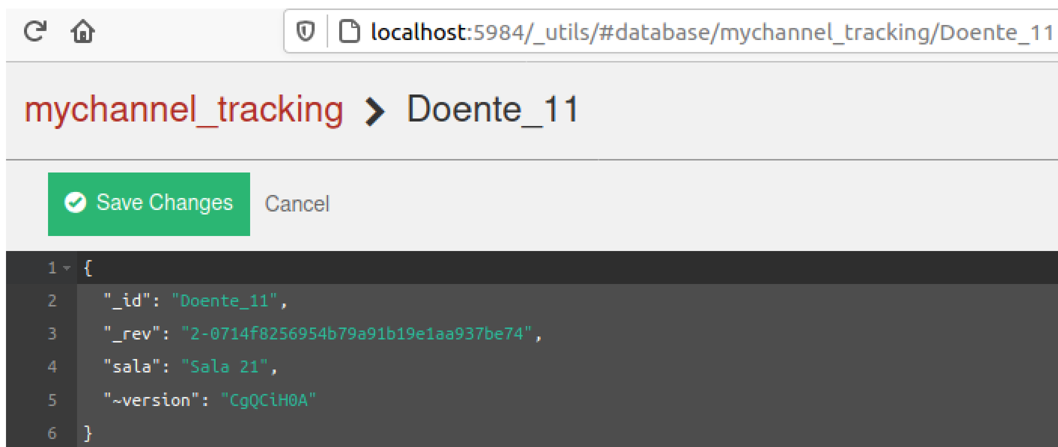


Figure 31 - Doente_11 asset in the database

6.3.9. Change Medical Device room property(changeDispMedSala)

The *changeDispMedSala* function allows changing the room (sala) property of any stored Medical Device (Dispositivo Médico) asset in the blockchain network. This function takes the two following arguments:

– Arg1: Medical Device ID (id)
– Arg2: Room (sala)

The first argument is used to identify the asset and the second argument is the property that will be changed.

The following image (Fig. 32) shows a POST request made to the API with the function *changeDispMedSala*:

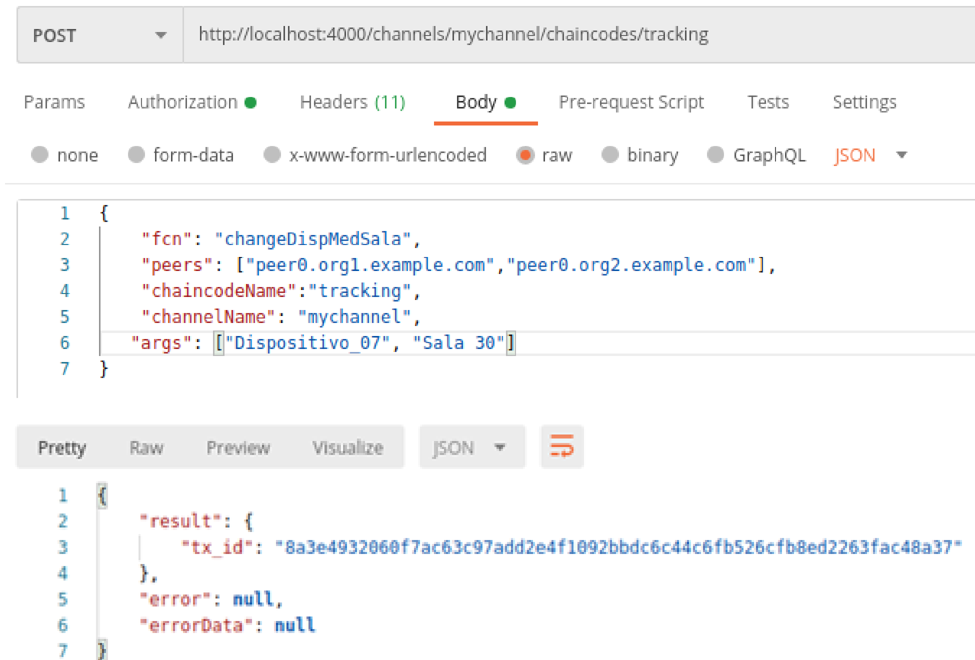


Figure 32 - changeDispMedSala function API call

6.3.10. Change Medical Device doctor property (changeDispMedMedico)

The *changeDispMedMedico* function allows changing the doctor (medico) property of any stored Medical Device (Dispositivo Médico) asset in the blockchain network. This function takes the two following arguments:

– Arg1: Medical Device ID (id)
– Arg2: Doctor ID (Medico ID)

The first argument is used to identify the asset and the second argument is the property that will be changed.

The following image (Fig. 33) shows a POST request made to the API with the function *changeDispMedMedico*:

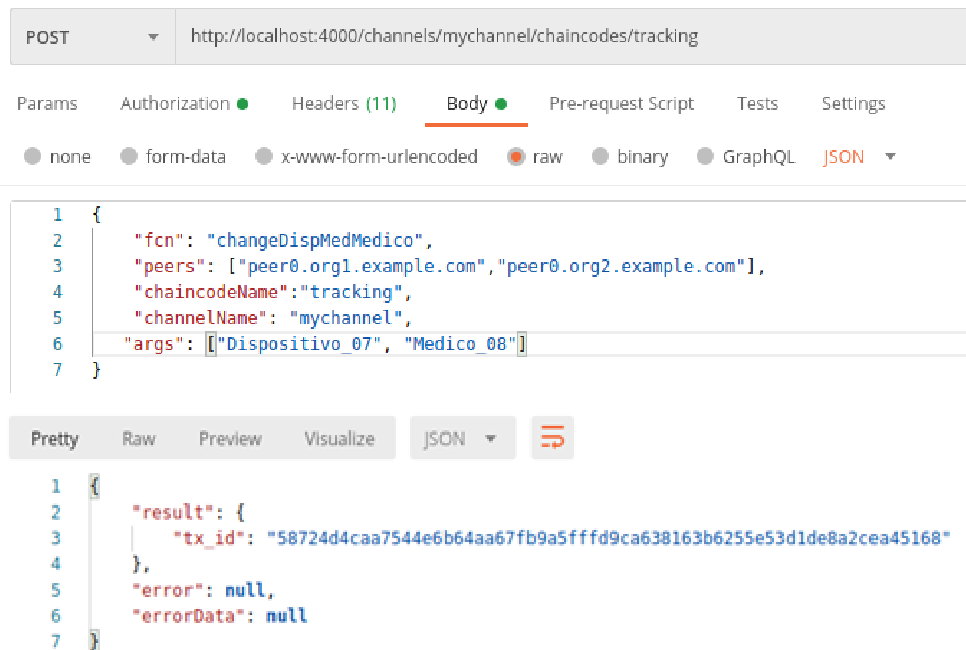


Figure 33 - changeDispMedMedico function API call

6.3.11. Change Medical Device patient property (changeDispMedDoente)

The *changeDispMedDoente* function allows changing the patient (doente) property of any stored Medical Device (Dispositivo Médico) asset in the blockchain network. This function takes the two following arguments:

- Arg1: Medical Device ID (id)
- Arg2: Patient ID (Doente ID)

The first argument is used to identify the asset and the second argument is the property that will be changed.

The following image (Fig. 34) shows a POST request made to the API with the function changeDispMedDoente.

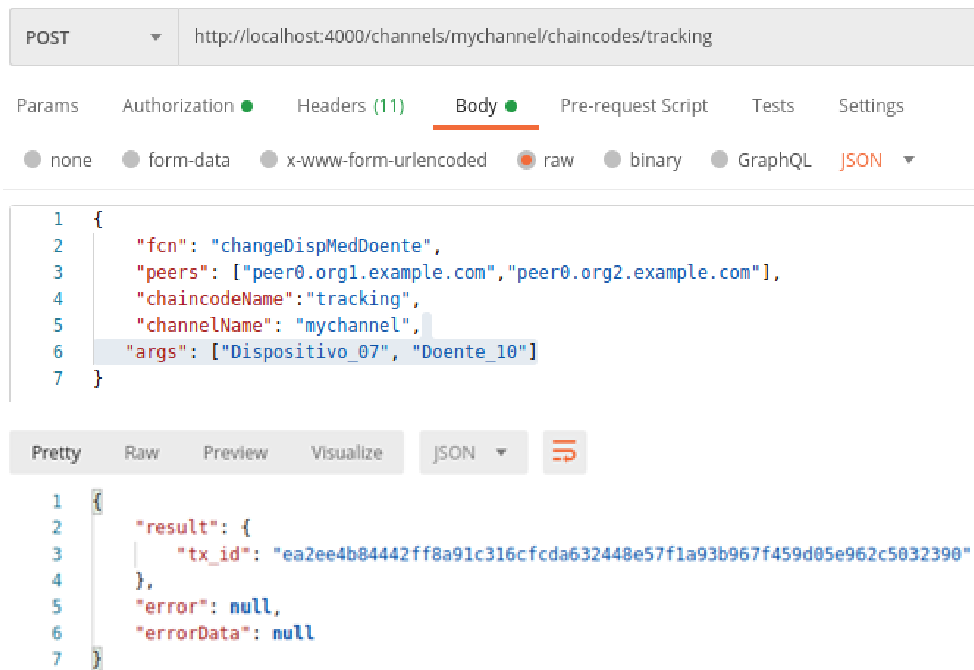


Figure 34 - changeDispMedDoente function API call

The following image (Fig. 35) represents the recently changed asset in the database.

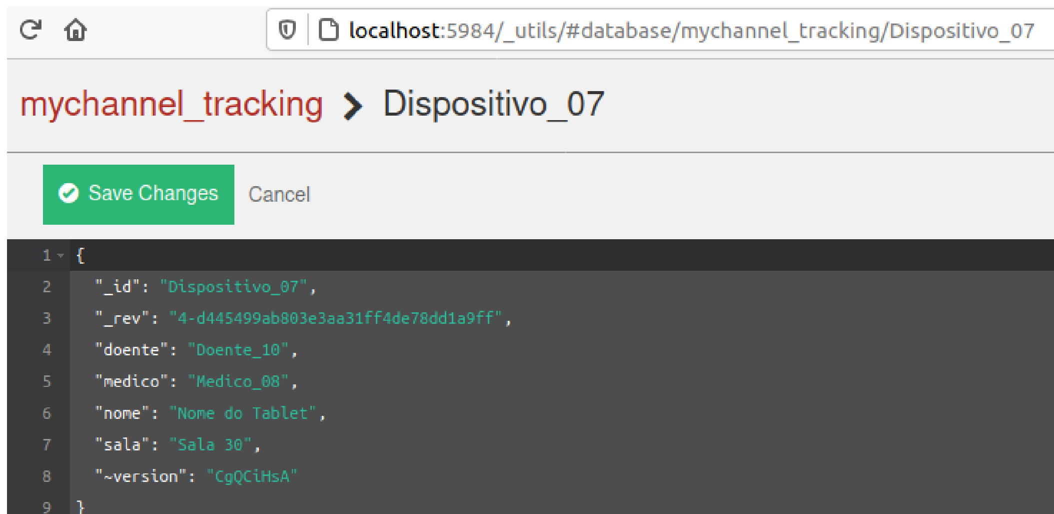


Figure 35 - Dispositivo_07 asset in the database

6.3.12. Get History of Asset (getHistoryForAsset)

As many assets are changed over time, there is a need to retrieve the information history of an asset. This function was created to satisfy those needs. The function returns the asset state over time as well as a timestamp and transaction id.

The function takes the following argument:

– Arg1: Asset ID (id)

The following image (Fig. 36) shows a GET request made to the API with the function *getHistoryForAsset* for the *Dispositivo_07* asset:

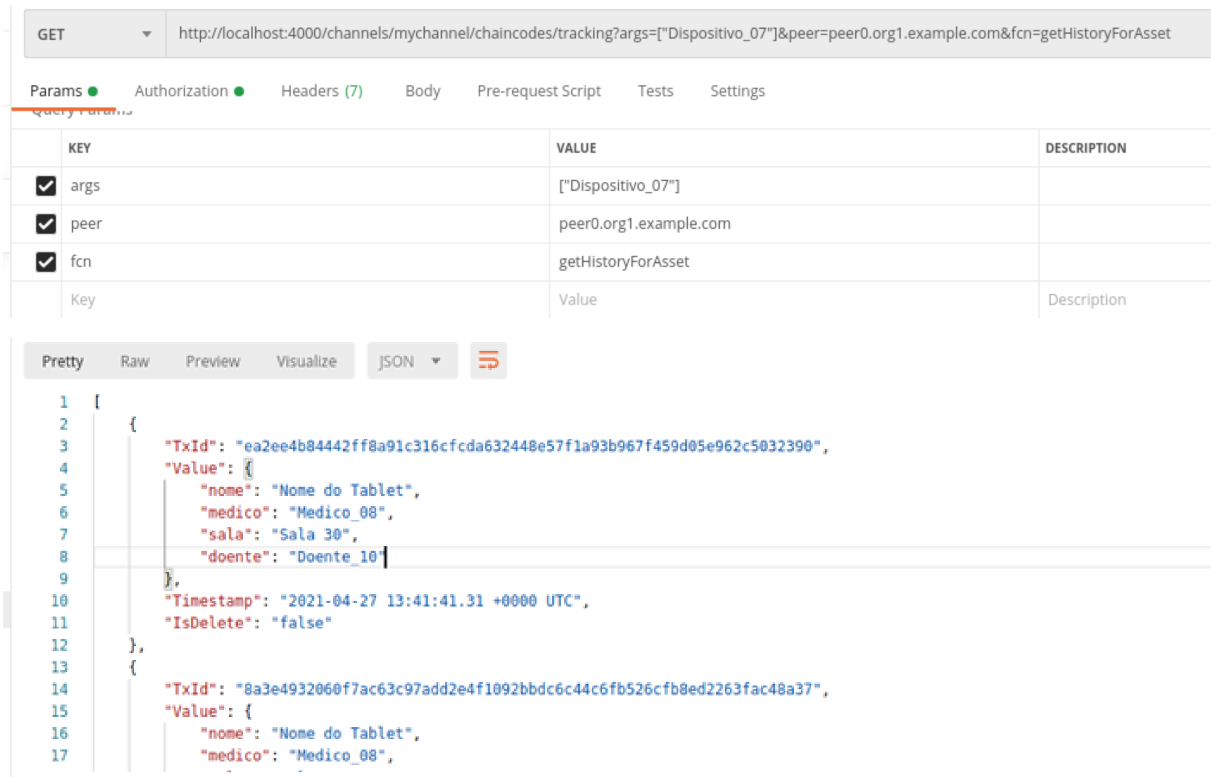


Figure 36 - getHistoryForAsset function API call

The following image (Fig. 37) shows a more detailed view of the response body, as well as the asset state status over time:

```
[
    {
        "TxId": "ea2ee4b84442ff8a91c316cfcda632448e57f1a93b967f459d05e962c5032390",
        "Value": {
            "nome": "Nome do Tablet",
            "medico": "Medico_08",
            "sala": "Sala 30",
            "doente": "Doente_10"
        },
        "Timestamp": "2021-04-27 13:41:41.31 +0000 UTC",
        "IsDelete": "false"
    },
    {
        "TxId": "8a3e4932060f7ac63c97add2e4f1092bbdc6c44c6fb526cfb8ed2263fac48a37",
        "Value": {
            "nome": "Nome do Tablet",
            "medico": "Medico_08",
            "sala": "Sala 30",
            "doente": "Doente_01"
        },
        "Timestamp": "2021-04-27 13:40:59.672 +0000 UTC",
        "IsDelete": "false"
    },
    {
        "TxId": "58724d4caa7544e6b64aa67fb9a5fffd9ca638163b6255e53d1de8a2cea45168",
        "Value": {
            "nome": "Nome do Tablet",
            "medico": "Medico_08",
            "sala": "Sala 1",
            "doente": "Doente_01"
        },
        "Timestamp": "2021-04-27 13:40:09.452 +0000 UTC",
        "IsDelete": "false"
    },
    {
        "TxId": "cc1918c1194cbb4acc56893a6b071f507f63066d7425444fc8a1beb39b07f113",
        "Value": {
            "nome": "Nome do Tablet",
            "medico": "Medico_01",
            "sala": "Sala 1",
            "doente": "Doente_01"
        },
        "Timestamp": "2021-04-27 13:00:31.49 +0000 UTC",
        "IsDelete": "false"
    }
]
```


Figure 37 - Detailed view of the getHistoryForAsset response body for the Dispositivo_07 asset

The asset history is sorted in inverse chronological order, in other words, from the most recent asset state to the oldest asset state.

The asset state is accompanied by a Transaction ID which points to the transaction responsible for the asset state change and a Timestamp in which the transaction occurred

By chronological the *Dispositivo_07* changed the *medico* property from *"Medico_01"* to *"Medico_08".* Then, the *sala* property changed from *"Sala 1"* to *"Sala 30"*. Finally, the *doente* property changed from *"Doente_01"* to *"Doente_10"*.

All of the described changes were used as an example in the previous chapters so this chapter confirms the results obtained on the previous chapters.

6.4. Blockchain Explorer

To develop the Blockchain Explorer part of the project a GitHub repository was downloaded using the following command (Pavan, n.d.-b):

- git clone https://github.com/adhavpavan/ContainerisingBlockchainExplorer.git

After cloning the repository, the *crypto-config* folder previously created in chapter 6.2 was copied to the *ContainerisingBlockchainExplorer* folder.

Inside the *connection-profile* folder there are two files:

- first-network.json
- first-network_2.2.json

For this project, the file *first-network_2.2.json* was used. In this folder, the correct paths to the previously copied crypto config certificates and private keys were provided. The correct paths to these files must be provided otherwise blockchain explorer initialization will fail with a wallet creation error. This file also contains the login credentials necessary for logging into Blockchain Explorer. These credentials can be changed in this file as well.

Under the main folder, in the *config.json* file, the path to the *first-network_2.2.json* was provided. The *.env* file contains the compose project name which, in this case, needs to be the same as the one used to set up the Hyperledger Fabric blockchain. In this case, the compose project name used is *"artifacts"*.

In the *docker-compose.yaml* file, many configurations influence the docker container behavior. Initially, nothing in this file was changed, but after chapter 6.6 there was a need to change the container ports because the 8080 port was already being used by *cadvisor* which is a Prometheus component. The port was changed to port 8000. Since the first port refers to the local machine and the second port points to the port used inside the container, only the first port was changed.

After everything is set up correctly, the following command was used inside the "/home/hugo/repo/ContainerisingBlockchainExplorer" folder to initialize the container:

– docker-compose up -d

When the command finishes executing Blockchain Explorer must be running and a login screen will be prompted to the user (Fig. 38).

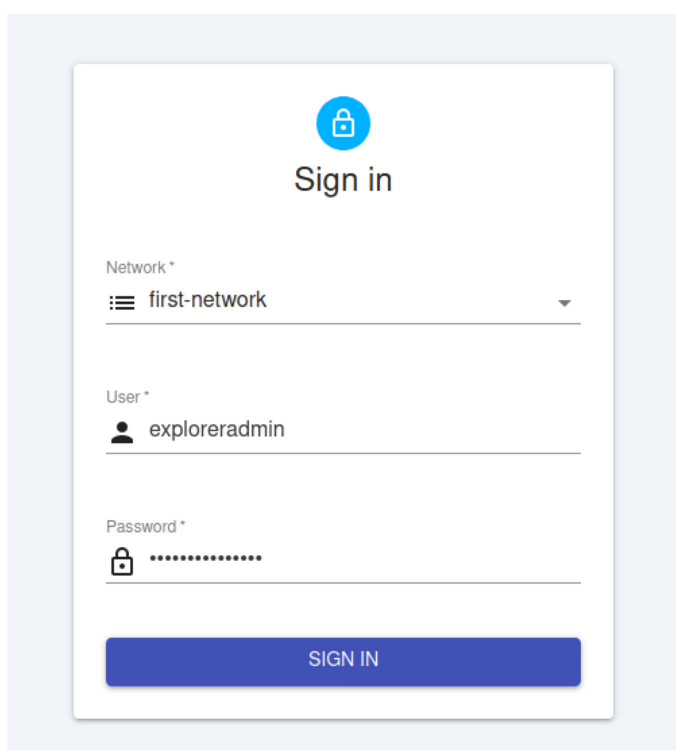


Figure 38 - Blockchain Explorer Login screen

After logging in, a dashboard about the blockchain is presented with many metrics that give a better overall view of the blockchain (Fig. 39).

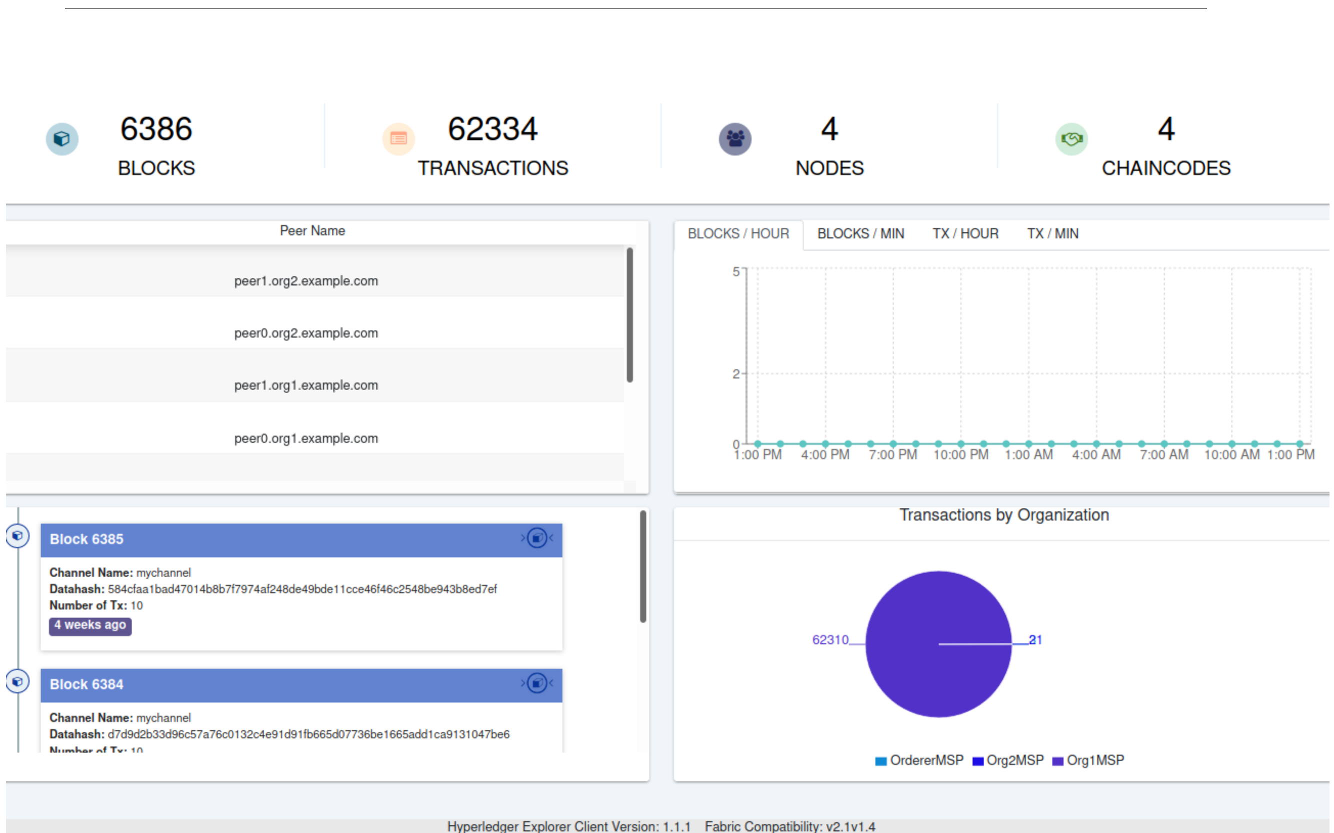


Figure 39 - Blockchain Explorer Dashboard

The dashboard provides the visualization of the peers available in the blockchain, the most recent blocks added, the transactions made by the organization, and the block throughput. Note that in the image above the maximum time scale is one hour, and the image was taken more than one hour after the transactions were made. This is why it shows that there are 0 blocks/hour.

On the top of the page, the tabs allow for a more detailed view of transactions, blocks, network participants, chaincodes and channels.

The following images (Fig. 40 to Fig 44) show examples of the level of detail of transactions, blocks, participants, channels, and chaincodes that Blockchain Explorer provides:

Transaction Details

Transaction ID:	7372408a230018d849bbe08f57f99399a1db89e7d654a3c7769702c8d59c6683
Validation Code:	
Payload Proposal Hash:	74b6679756916edbe00607373dfc4605498237bf940ea3976075e3bdb7b36038
Creator MSP:	Org1MSP
Endoser:	{"Org1MSP","Org2MSP"}
Chaincode Name:	tracking
Type:	ENDORSER_TRANSACTION
Time:	2021-03-31T19:16:31.364Z
Reads:	▼ root: [] 2 items ▶ 0: {} 2 keys ▶ 1: {} 2 keys
Writes:	▼ root: [] 2 items ▶ 0: {} 2 keys ▶ 1: {} 2 keys

Figure 40 - Blockchain Explorer Transaction details

Block Details

Channel name:	mychannel
Block Number	6385
Created at	2021-03-31T19:16:31.364Z
Number of Transactions	10
Block Hash	44fd058bf5131074bbafc6bee2680d2fd0f952642f45e5793060e5fc6cb7ad6a
Data Hash	584cfaa1bad47014b8b7f7974af248de49bde11cce46f46c2548be943b8ed7ef
Prehash	97cd1ea21aa13a995e5b0cc7a5c5fbadc372b5e1040ba9a635868c00c12b278c

Figure 41 - Blockchain Explorer Block details

Chaincode Name	Channel Name	Path	Transaction Count	Versior
tracking	mychannel	-	59180	4
tracking	mychannel	-	59177	5
tracking	mychannel	-	59202	2
tracking	mychannel	-	59193	3

Figure 42 - Blockchain Explorer Chaincode details

ID	Channel Name	Blocks	Transactions	Timestamp
3	mychannel	6386	62334	2021-03-16T22:05:18.000Z

Figure 43 - Blockchain Explorer channels

				Ledger Height		
Peer Name	Request Url	Peer Type	MSPID	High	Low	Unsigned
peer1.org2.example.c...	peer1.org2.example.c...	PEER	Org2MSP	0	34032	true
peer0.org2.example.c...	peer0.org2.example.c...	PEER	Org2MSP	0	34032	true
peer1.org1.example.c...	peer1.org1.example.c...	PEER	Org1MSP	0	34032	true
peer0.org1.example.c...	peer0.org1.example.c...	PEER	Org1MSP	0	34032	true
orderer.example.com	orderer.example.com:...	ORDERER	OrdererMSP	-	-	-
orderer2.example.com	orderer2.example.com...	ORDERER	OrdererMSP	-	-	-
orderer3.example.com	orderer3.example.com...	ORDERER	OrdererMSP	-	-	-

Previous Page 1 of 1 100 rows Next

Figure 44 - Blockchain Explorer network participants

6.5. Hyperledger Caliper

To develop the Hyperledger Caliper part of the project a GitHub repository was downloaded using the following command (Pavan, n.d.-c):

- git clone https://github.com/adhavpavan/ContainerisingCaliperForFabricBenchmark.git

After cloning the repository, the *crypto-config* folder previously created in chapter 6.2 available in the path "/BasicNetwork-2.0/artifacts/channel/crypto-config" was copied to the following path:

- "ContainerisingCaliperForFabricBenchmark/caliper-benchmarks-local/networks/fabric/pavan-v2.1/"

Inside the folder to which the *crypto-config* was copied, there is a file called *network-config_2.2.yaml*. In this file, the path indicated should match with the certificates and private keys stored in the recently copied *crypto-config* folder.

The smart contract folder containing the *tracking.go* file previously created in chapter 6.2 was also copied to the "ContainerisingCaliperForFabricBenchmark" folder in the following path:

- “ContainerisingCaliperForFabricBenchmark/caliper-benchmarks-local/src/fabric/samples/fabcar/go”

To test the functions created in the smart contract, one test function was created per function in the smart contract in the following path:

- “/ContainerisingCaliperForFabricBenchmark/caliper-benchmarks-local/benchmarks/scenario/simple/pavan-v2.2”

These functions are organized in separate javascript files and each function is responsible for generating random assets that Caliper uses to insert into the blockchain in order to benchmark the network.

The following image (Fig. 45) shows one of the functions created:

```
1   'use strict';
2
3   const { WorkloadModuleBase } = require('@hyperledger/caliper-core');
4
5   const salas = ['Sala 1', 'Sala 2', 'Sala 3', 'Sala 4', 'Sala 5', 'Sala 6', 'Sala 7', 'Sala 8', 'Sala 9', 'Sala 10'];
6   const nomes = ['Tablet 1', 'Tablet 2', 'Tablet 3', 'Tablet 4', 'Tablet 5', 'Tablet 6', 'Tablet 7', 'Tablet 8', 'Tablet 9', 'Tablet 10'];
7
8   /**
9    * Workload module for the benchmark round.
10   */
11  class CreateDispMedWorkload extends WorkloadModuleBase {
12      /**
13       * Initializes the workload module instance.
14       */
15      constructor() {
16          super();
17          this.txIndex = 0;
18      }
19      /**
20       * Assemble TXs for the round.
21       * @return {Promise<TxStatus[]>}
22       */
23      async submitTransaction() {
24          this.txIndex++;
25          let id = 'Client' + this.workerIndex + '_Dispositivo' + this.txIndex.toString();
26          let nome = nomes[Math.floor(Math.random() * nomes.length)];
27          let medico = 'Client' + this.workerIndex + '_Medico' + this.txIndex.toString();
28          let sala = salas[Math.floor(Math.random() * salas.length)];
29          let doente = 'Client' + this.workerIndex + '_Doente' + this.txIndex.toString();
30
31          let args = {
32              contractId: 'tracking',
33              contractVersion: 'v5',
34              contractFunction: 'createDispMedico',
35              contractArguments: [id, nome, medico, sala, doente],
36              timeout: 30
37          };
38
39          await this.sutAdapter.sendRequests(args);
40      }
41  }
42
43  /**
44   * Create a new instance of the workload module.
45   * @return {WorkloadModuleInterface}
46   */
47  function createWorkloadModule() {
48      return new CreateDispMedWorkload();
49  }
50
51  module.exports.createWorkloadModule = createWorkloadModule;
```

Figure 45 - Hyperledger Caliper CreateDispMedico function for random asset generation

After creating the functions, the *config.yaml* file was edited to implement the recently created functions. In this file, the following parameters were specified:

- label: Name of the function that will appear in the report;
- txNumber: Number of transactions that will be sent into the network;
- rateControl – type: Type of rate;
- opts – tps: Transactions per second;
- workload – module: Path of the javascript file that generates the assets;

Throughout the benchmark phase, this configuration was constantly changed in order to create new benchmark scenarios and compare them.

To execute caliper, the following command was used in the main folder:

- docker-compose up -d

To follow the Caliper process inside the docker container, the following command was used:

- docker logs caliper_2.2 -f

Note that in this project, the Caliper container name is *caliper_2.2*. The container name should match the name used in the *docker-compose.yaml* file. To verify the docker containers of the system currently running the following command was used:

- docker ps

After executing successfully, Hyperledger Caliper will generate a report based on the configurations used and the functions created. In each report, multiple metrics indicate the performance of each function of the blockchain:

- Succc - How many transactions were successful;
- Fail – How many transactions were unsuccessful;
- Send Rate (TPS) – How many transactions are sent in a second;

– Max Latency (s) – Maximum read time in seconds of the set of transactions;

– Min Latency (s) – Minimum read time in seconds of the set of transactions;

– Avg Latency (s) – Average read time in seconds of the set of transactions;

– Throughput (TPS) - How many transactions are received in a second;

The following image (Fig. 46) shows an example of a report generated with Hyperledger Caliper in the course of the project:

Caliper report

Summary of performance metrics

Name	Succ	Fail	Send Rate (TPS)	Max Latency (s)	Min Latency (s)	Avg Latency (s)	Throughput (TPS)
Create Beacon	961	39	428.6	17.42	5.07	11.09	55.2
Create Medico	997	3	527.1	16.95	3.09	10.56	57.6
Create Doente	995	5	484.0	18.24	2.90	11.31	53.4
Create Dispositivo Medico	997	3	500.0	17.94	3.71	12.12	53.3
Change Sala property of Beacon	1000	0	510.2	19.11	5.15	13.60	51.0
Change Sala property of Doente	1000	0	532.2	18.94	4.92	13.79	50.3
Change Medico property of Dispositivo Medico	1000	0	570.1	19.19	3.09	14.01	50.0
Change Doente property of Dispositivo Medico	1000	0	585.8	20.25	2.87	13.64	48.6
Get History of Asset	987	13	460.2	16.89	4.86	11.91	56.4

Figure 46 - Hyperledger Caliper report

6.6. Prometheus and Grafana

To develop the Prometheus and Grafana part of the project a GitHub repository was downloaded using the following command (Pavan, n.d.-c):

– git clone https://github.com/hyperledger/caliper-benchmarks.git

After cloning the repository, the folder "/networks/Prometheus-grafana" was copied to the already existing folder "ContainerisingCaliperForFabricBenchmark/caliper-benchmarks-local/networks/" in order to integrate Prometheus and Grafana with Hyperledger Caliper. The following image (Fig. 47) shows the folder structure obtained with the copied folder highlighted in blue:

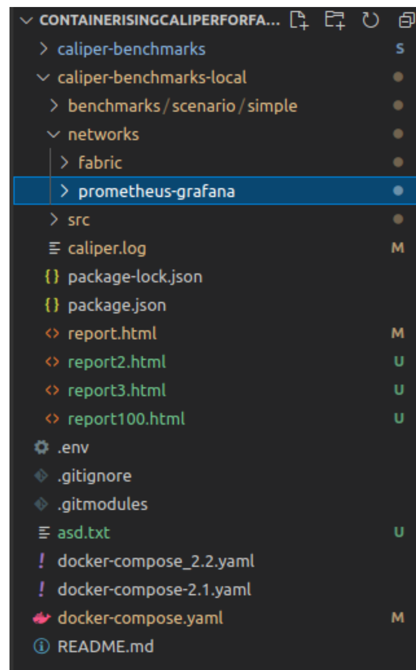


Figure 47 - Prometheus folder structure

Inside the "prometheus-grafana" folder there are two yaml files. The name of the "docker-compose-bare.yaml" file was changed to "docker-compose.yaml" for docker to recognize which file to use when the container gets initialized.

Inside the "docker-compose.yaml" reside all the container configurations necessary for docker including the ports in which Prometheus and Grafana are going to run. The ports used were the following:

- Grafana : ports - 3000:3000
- Prometheus: ports - 9090:9090

These ports must be available otherwise initializing the container may fail. If a port is already in use simply change the first port to an available one in the local machine.

After everything is set up correctly, the following command was used inside the "/home/hugo/repo/ContainerisingCaliperForFabricBenchmark/caliper-benchmarks-local/networks/prometheus-grafana" folder to initialize the containers:

- docker-compose up -d

When the command finishes executing both Prometheus and Grafana must be running. In this project, Prometheus is available on "http://localhost:9090" and Grafana is available on "http://localhost:3000".

The following images (Fig. 48 and Fig. 49) show the results obtained:

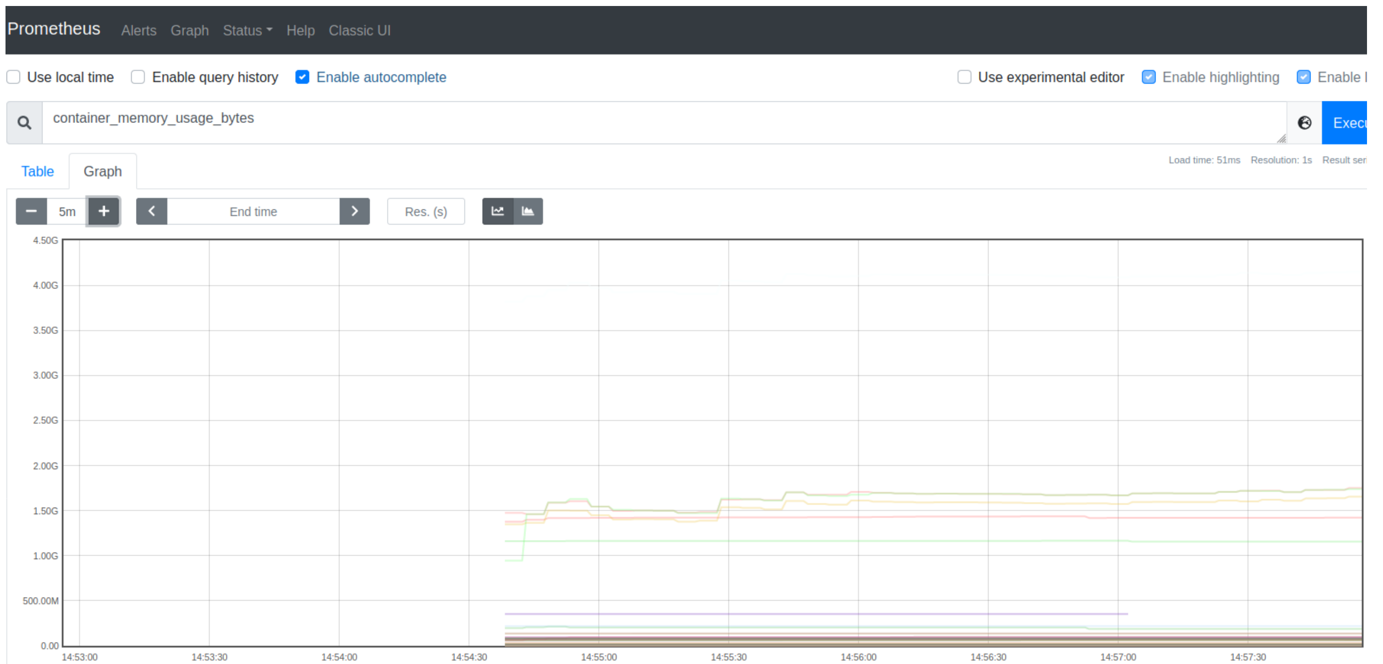


Figure 48 – Prometheus

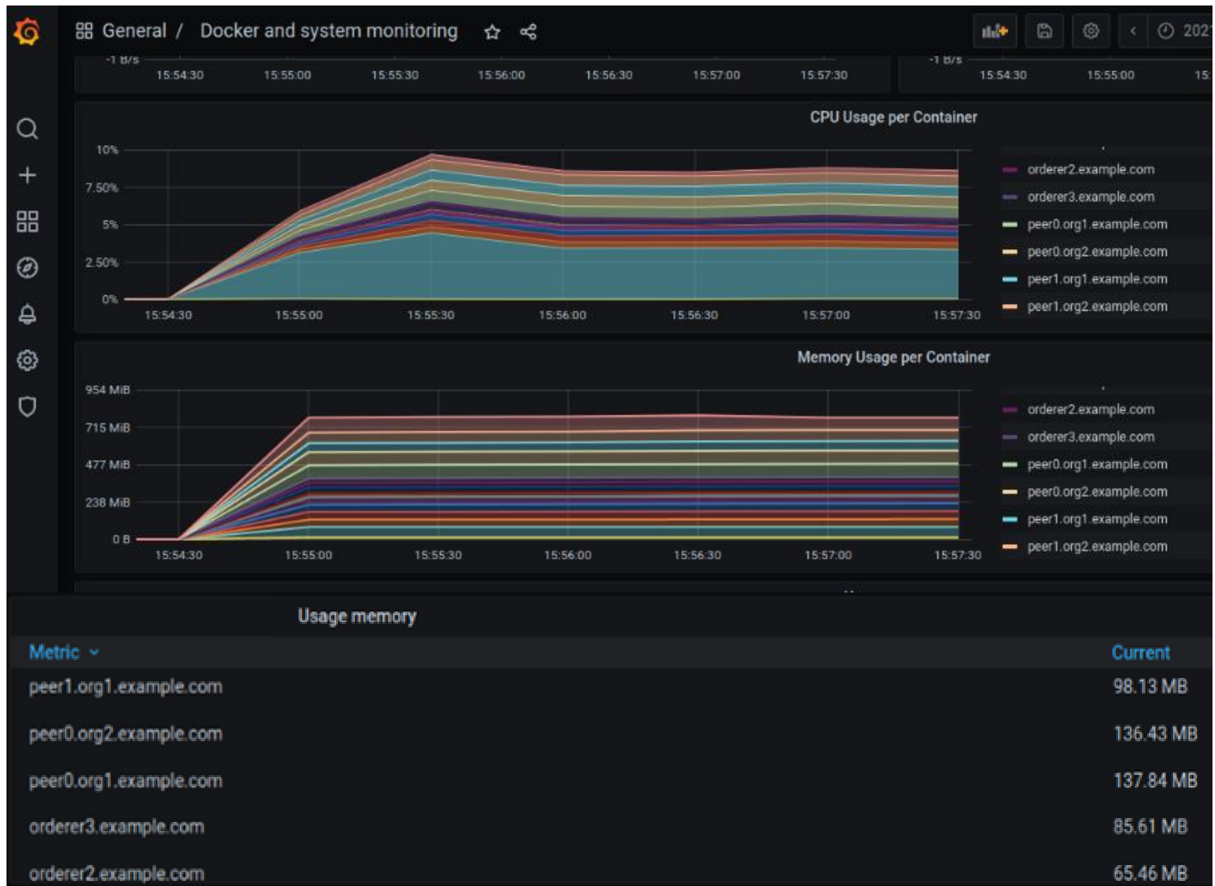


Figure 49 – Grafana

7. Benchmarking

This chapter describes the benchmark methodology used in this project as well as all the benchmarks that were conducted and the respective graphs. A brief analysis of the graphs will be presented.

Several experiments were carried out to analyze the performance of the proposed blockchain solution, namely for analyzing the transaction latencies as well as success rates for 1k transactions and 10k transactions at 40 transactions per second (TPS), and for 100 transactions at 100 TPS, 1k TPS, and 10k TPS. A benchmarking tool is essential to conduct these types of experiments. Hyperledger Caliper was used for this very purpose. Hyperledger Caliper is a blockchain benchmarking tool that allows users to assess a blockchain implementation's performance against a set of predefined use cases.

In the first experiment, two benchmarks were carried out, with a send rate of 40 transactions per second. In the first, 1000 transactions for each function created were committed in the network in order to evaluate each function's performance. In the second benchmark, 10000 transactions for each function were made. Note that the first function, Create Beacon, is missing, this is due to the fact that the first transaction of the benchmark takes substantially more time than the rest of the transactions making the variation between function metrics harder to visualize. However, the first function performance has the same behavior as the other functions for both benchmarks.

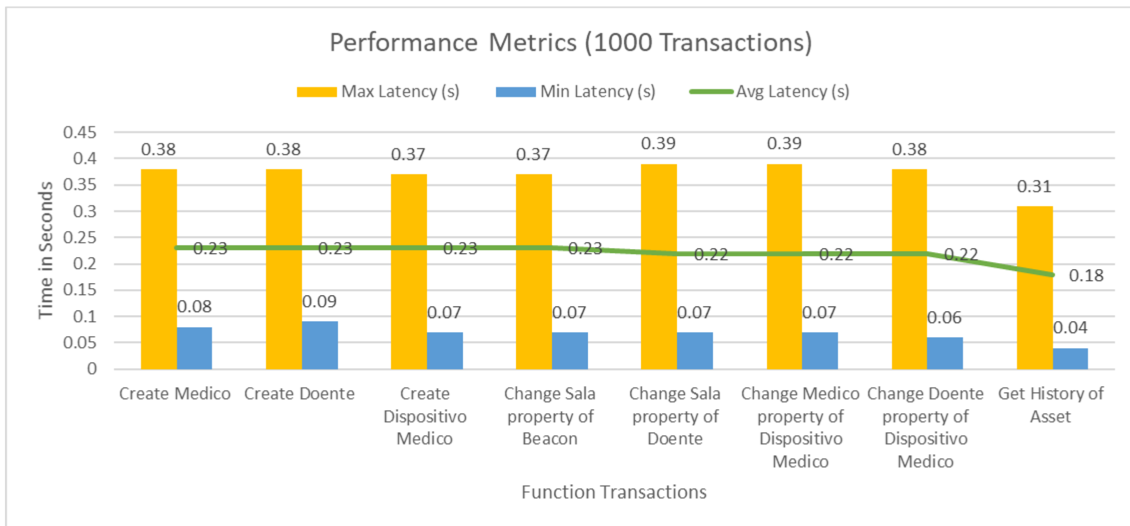


Figure 50 - Performance Metrics (1k transactions per function)

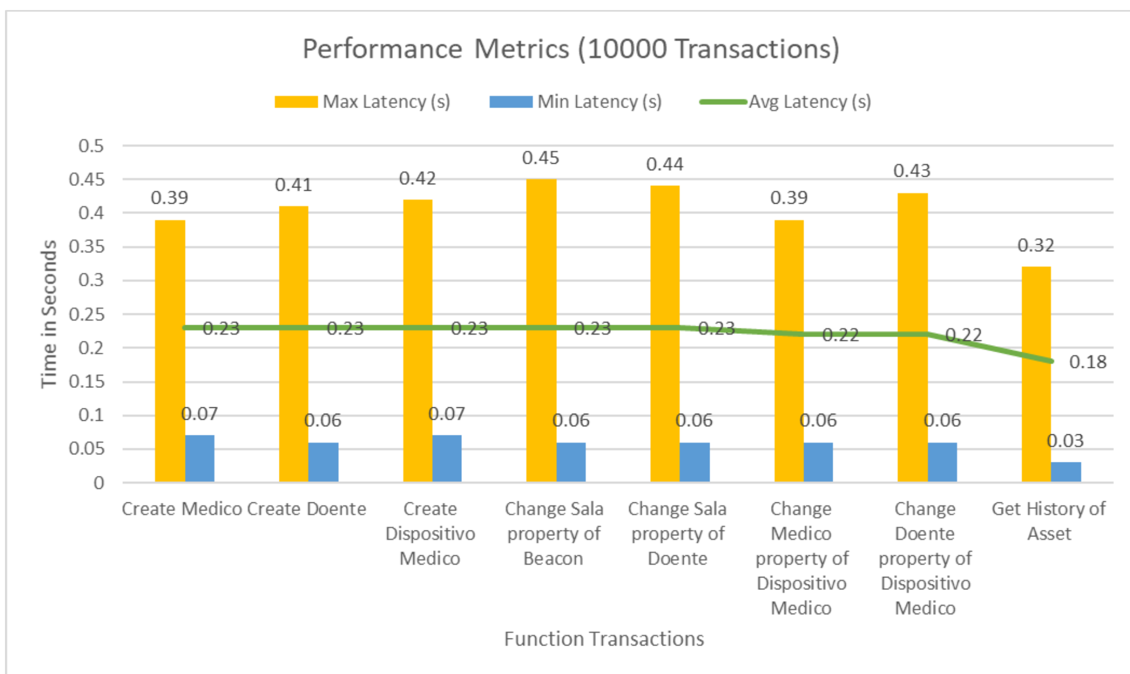


Figure 51 - Performance Metrics (10k transactiosn per function)

Analyzing Fig 50 and Fig 51, we can see that moving from 1k transactions to 10k transactions had a slight increase in the transaction's maximum latency, but the average latency stayed almost the same. This probably happens since the second benchmark environment applies more transactions, so there is a bigger chance of one of those transactions taking more time, hence increasing the max latency value. It can be stated that the number of transactions has almost no impact on blockchain performance. Get History of Asset is a query function and is significantly faster than its counterparts, having recorded the lowest maximum, minimum, and average latency From this result, it can be said that query functions are overall faster in terms of latency than the asset creation and asset state change functions.

For the second experiment, three benchmarks were carried out for 100 transactions per function with 100, 1k, and 10k TPS respectively to analyze the Send Rate and Throughput of the proposed solution. After concluding the three benchmarks, since Hyperledger Caliper outputs the send rate and throughput of every single function, we calculated the average of both metrics in order to have a better view of the performance, thus reaching a more significant conclusion. Fig 9 illustrates this analysis.

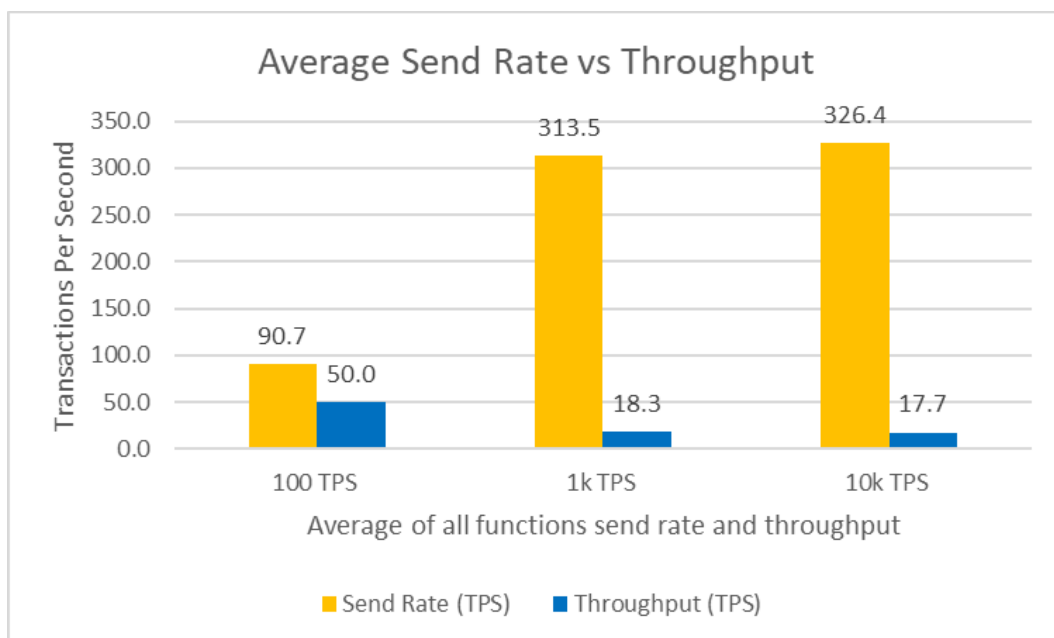


Figure 52- Average Send Rate vs Throughput

Analyzing Fig 52, we can see that the average sending rate increases with the TPS increase which is expected. Furthermore, the throughput decreased significantly with the increase of TPS.

This happens because the blockchain system cannot handle the high number of transactions being sent in a second and fails to commit the transaction. Fig 53 illustrates the average of transactions that failed per benchmark, which justifies the previous statement.

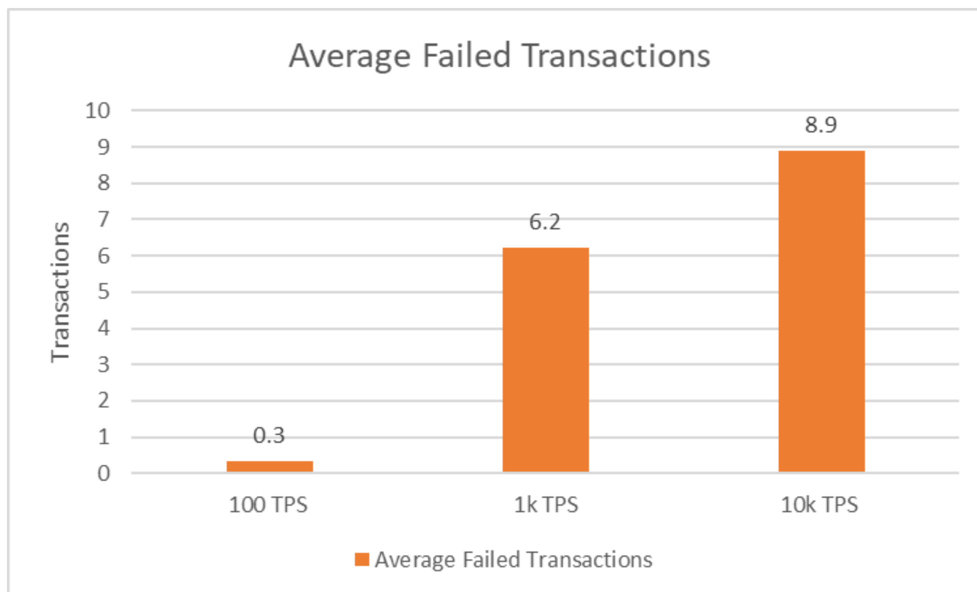


Figure 53 - Average Failed Transactions

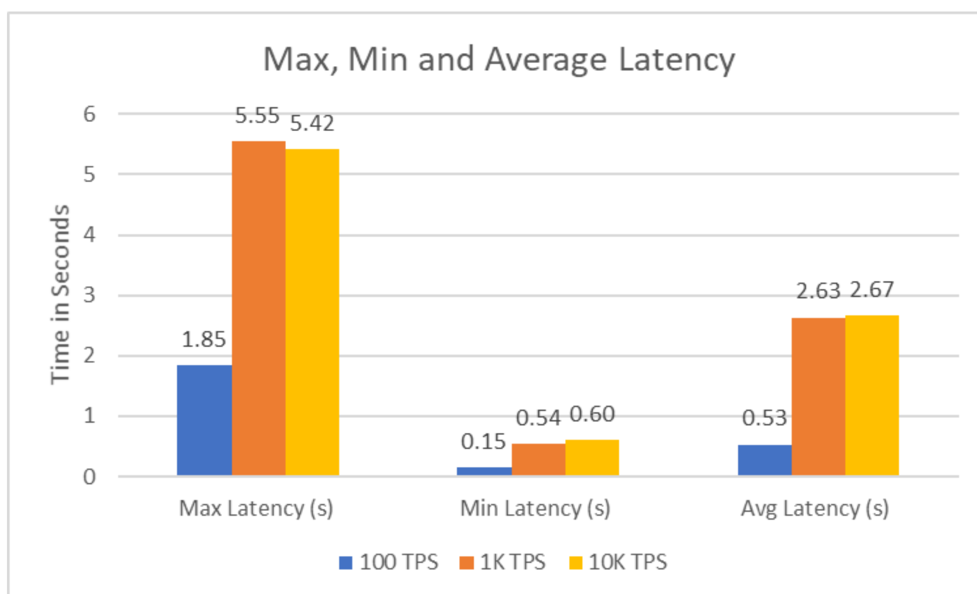


Figure 54 - Max, Min and Average Latency for 100, 1k and 10K TPS

As we can see in Fig 54, the average latency increases substantially from 100 to 1k TPS and a very slight increase from 1k to 10k TPS.

Note that the computers used to run these types of blockchain systems usually have up to 32 CPU cores per node and true multiprocessing. The machine used for this prototype is not ideal for this purpose, thus obtaining this kind of result.

The scalability of the system is directly impacted by the number of transactions sent to it and the processing power of the machine running the system.

In order to determine the blockchain network performance from the perspective of computational capacity, Prometheus was used to capture system metrics such as CPU usage and Memory usage, from the running docker containers.

During one of the previous benchmarks, Prometheus captured the results represented in the following graphics. Note that there was only one system used for this test, which results in having only one machine running both the blockchain network and the benchmark. This means that the system resources were shared across all the docker containers, which ultimately influences the final result. This becomes evident in the following graphic:

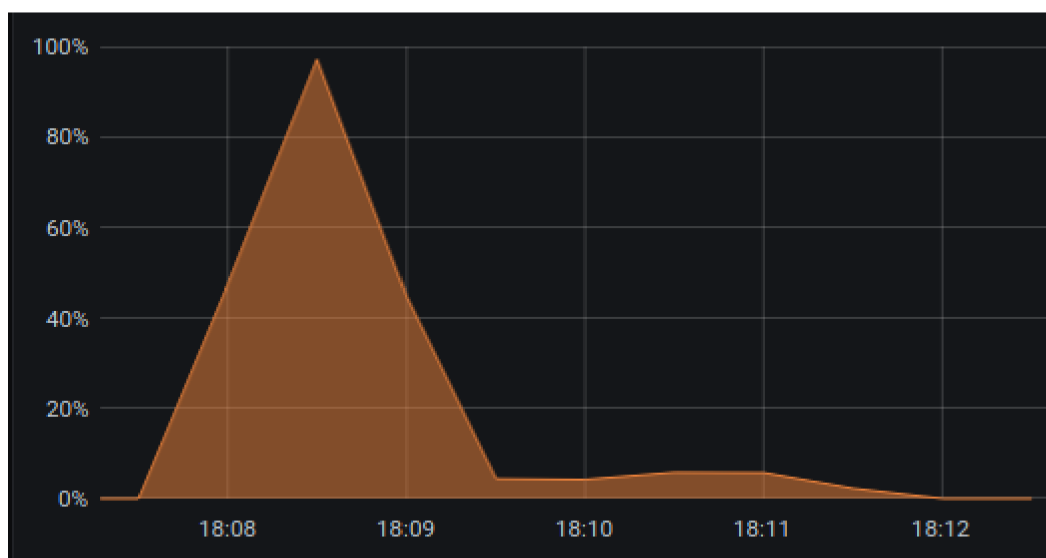


Figure 55 - Caliper container CPU usage during a benchmark

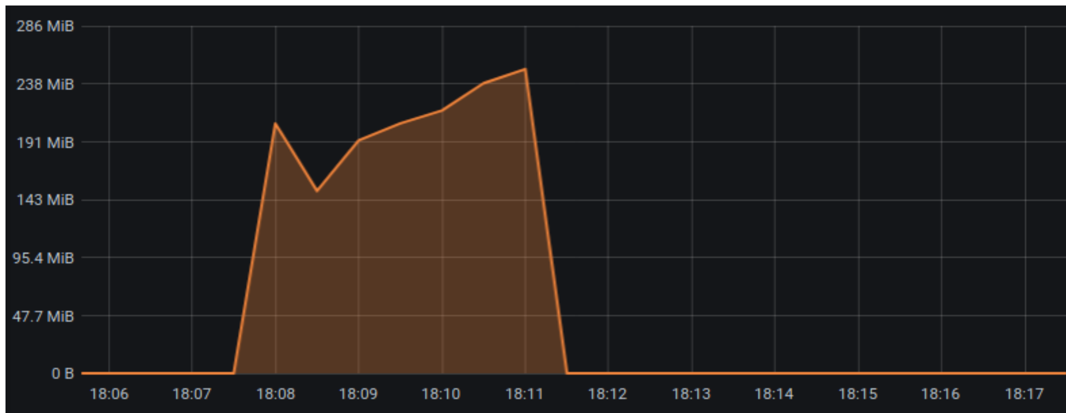


Figure 56 - Caliper container memory usage during a benchmark

As we can see in Fig 55 and Fig. 56, the caliper container, which is responsible for running the benchmark, has a CPU usage spike at around 97% usage and used around 200MB of memory on average. This has an impact on blockchain performance since the resources allocated for the caliper container cannot be used for the blockchain-related containers, slowing down the blockchain network even more.

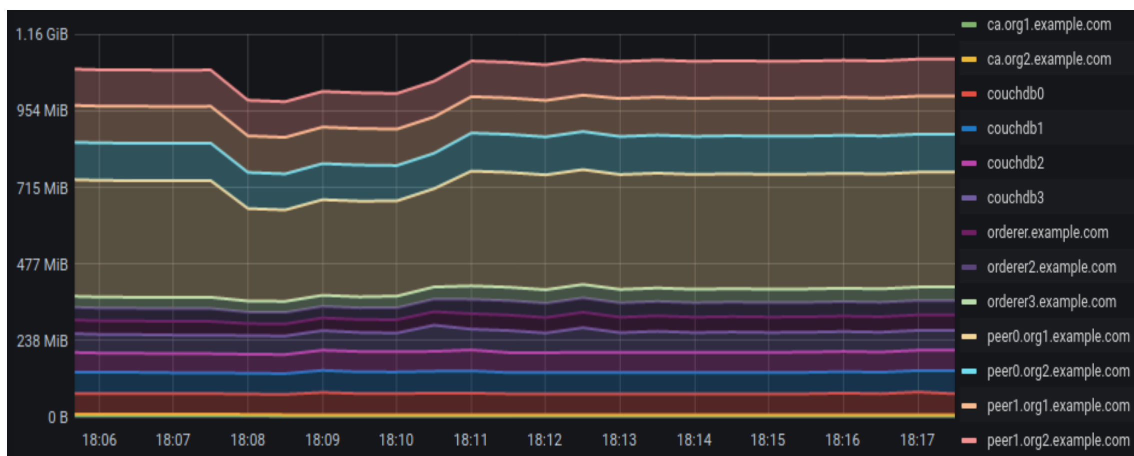


Figure 57 - All blockchain related containers accumulated memory usage

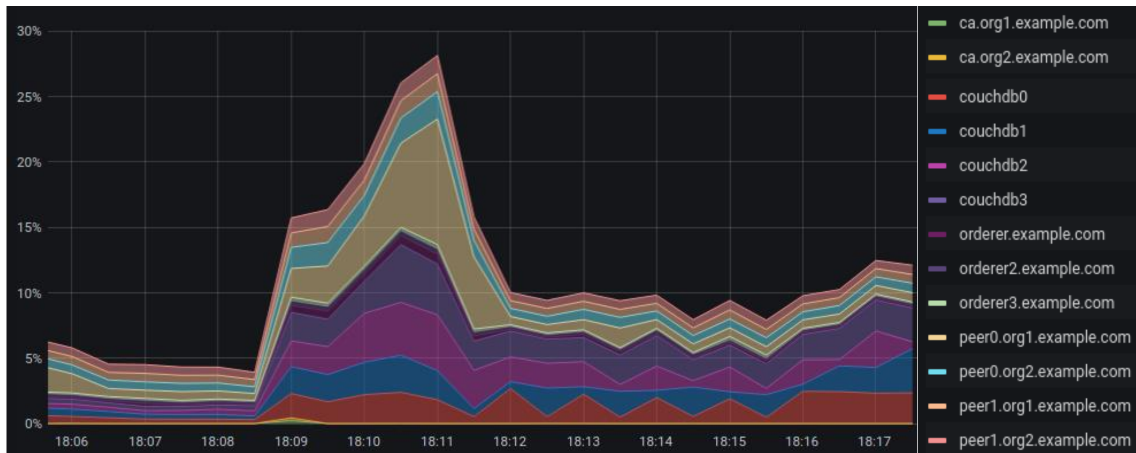


Figure 58 - All blockchain related containers accumulated CPU usage

As we can see in Fig 57 and Fig 58, the accumulated blockchain CPU usage peaked at 31%, and the memory consumption of all the blockchain-related containers was 1GB. The analysis of Fig 57 and Fig. 58 allows us to conclude that the resource consumption is directly proportional to the number of peers existent in the blockchain network. Note that the system will adapt to low resource availability and slow down. If given more resources the CPU usage and Memory usage could be significantly higher and the blockchain benchmark significantly faster as well.

The following graphics (Fig 59 to Fig 62) represent the two peers of an organization. As Peer 0 was used for the API calls, it consumed a substantially higher amount of memory and CPU than its counterpart. Note that during the benchmark, the memory usage decreases when CPU usage increases, and increases over again when CPU usage decreases. This probably occurs because the caliper container which is responsible for conducting the benchmark has priority over the peer containers, and consumes the memory resources since they are very limited in this system. An ideal benchmark environment would require the benchmarks and blockchain network to be running on different machines so that the resources are not shared and limited. Usually, blockchain systems run in server clusters that have much higher computational capability than the virtual machine used for these tests.

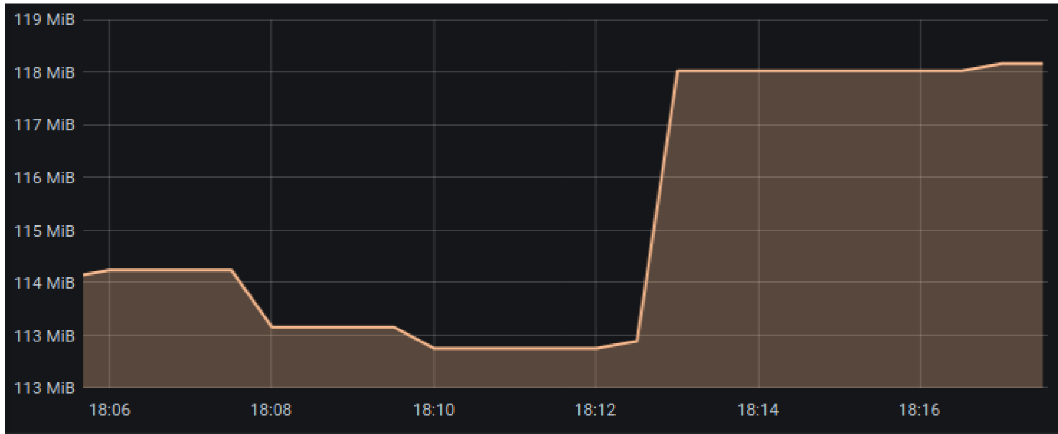


Figure 59 - Peer 1 Org 1 memory usage

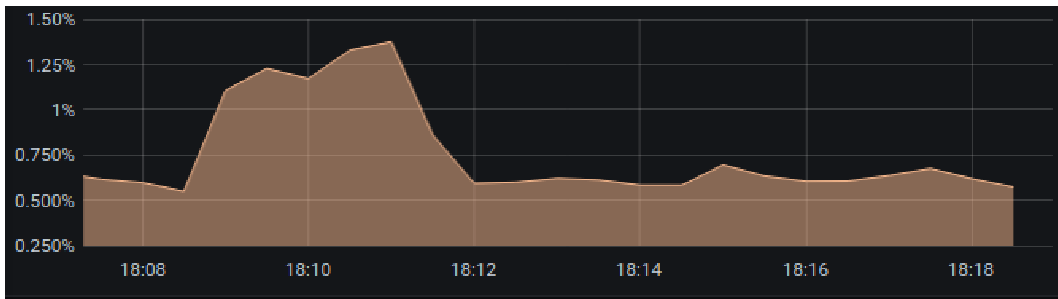


Figure 60 - Peer 1 Org 1 CPU usage

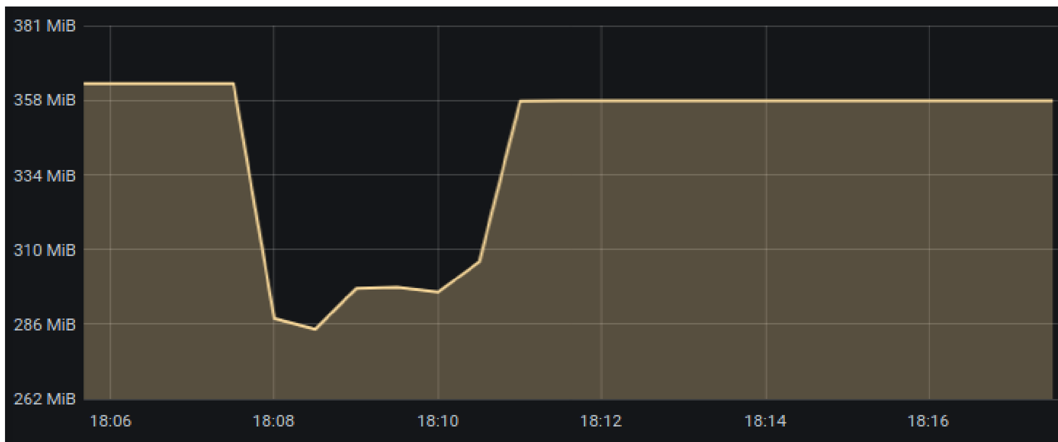


Figure 61 - Peer 0 Org 1 memory usage

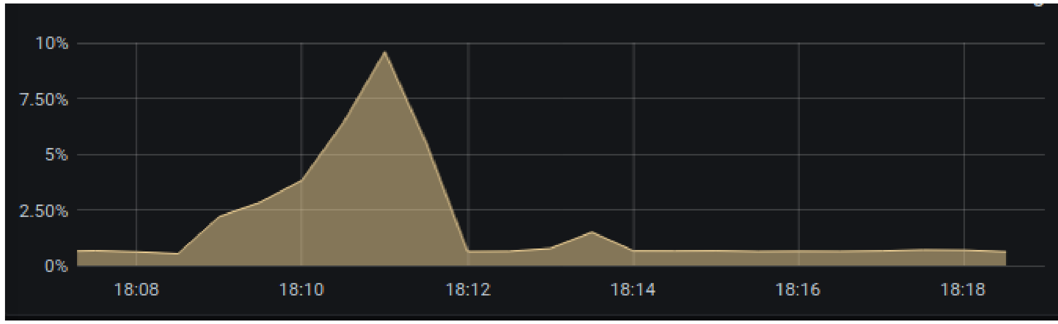


Figure 62 - Peer 0 Org 1 CPU usage

8. Conclusion

This dissertation was carried out in order to develop a Blockchain Network and a Rest API in the healthcare context. Initially, a search about blockchain technology was conducted to collect information about the intricacies involving blockchain operation, as well as its applications in the healthcare context. The result of this search aims to select the more efficient blockchain frameworks to be used in the implementation phase of the research. As a result, the framework used was Hyperledger Fabric as it is an open-source project which offers all the necessary tools to deploy a private blockchain. Furthermore, a search was conducted to find which tools for blockchain benchmarking were the most appropriate. Consequently, both Hyperledger Caliper and Prometheus were selected to be used in the benchmarking phase of the research. The former was chosen since it offers powerful metrics, it's easy to set up, and is developed by Hyperledger, which makes integration with Hyperledger Fabric easy and seamless. Moreover, another search that aims to find which tools for blockchain visualization were appropriate, was carried out. As a result, Blockchain Explorer was chosen for its simple but effective user interface that allows users to have a general view of the network, as well as see all the blocks and transactions made.

Subsequently, the implementation phase was conducted by installing and deploying Hyperledger Fabric and the programming of the Smart Contracts that reflect the data structure and logic of the project. After that, tools for blockchain benchmarking and tools for blockchain visualization were installed and configured and small tests and changes were made to perfect the network. For instance, the Smart Contract was developed in different phases to accommodate changes that allowed the best functioning of the network. Furthermore, benchmarks were defined and subsequently carried out in order to generate some graphics that allowed the analysis of the performance of the blockchain network. These benchmarks were defined in two distinct ways: a) the performance inherent to the blockchain network, such as average latency, throughput, etc; b) the computational performance of the system running the network such as CPU usage, memory usage, etc.

The benchmark analysis provided the following results: it can be stated that the number of transactions has almost no impact on blockchain performance; query functions are overall faster in terms of latency than the asset creation and asset state change functions; average sending rate increases with the TPS increase; the blockchain system cannot handle the high number of transactions being sent in a second and fails to commit the transaction; the scalability of the system

is directly impacted by the number of transactions sent to it and the processing power of the machine running the system; the resources allocated for the caliper container cannot be used for the blockchain-related containers, slowing down the blockchain network even more; the peer used for the API call uses significantly more resources than the other peers; that the resource consumption is directly proportional to the number of peers existent in the blockchain network.

To conclude, as it was previously stated, the objectives of this project were accomplished and allowed to develop a functional solution featuring Web API's and Blockchain.

The final solution provides a tamper-proof and immutable way of storing transactions which is essential for the Healthcare environment since data veracity must be achieved. Patients can be tracked through beacons and medical devices which originates an immutable medical history of the patient. For instance, which rooms have the patient been hospitalized in, and by which doctor he has been consulted.

Answering the Investigation Question, it can be stated that, by implementing a Blockchain Network, with well-designed Smart Contracts, which features a REST API that facilitates the communication between the Blockchain Network and other software, we ensure a tamper-proof, immutable, controlled, and secured way of storing data in the Healthcare context, given Blockchain's inherent characteristics (as it was explained in the state of the art).

For future work, we recommend deploying the developed blockchain network in a cluster to have a more realistic approach. On one hand, this would create more realistic results in the benchmarking and, on the other hand, would allow for a wider network since the computational power provided by the cluster is much greater than a virtual machine in a work computer. Running a blockchain network in a virtual machine hosted on a work computer is a major conditioning factor. Furthermore, the development of an android application with a seamless user interface to be deployed in medical devices that would interact with the currently developed API, which would complement the project, bringing broad benefits to the Healthcare business.

9. References

Ampel, B., Patton, M., & Chen, H. (2019). Performance modeling of hyperledger sawtooth blockchain. *2019 IEEE International Conference on Intelligence and Security Informatics, ISI 2019*, 59–61. https://doi.org/10.1109/ISI.2019.8823238

Baliga, A. (2017). Understanding Blockchain Consensus Models. *Whitepaper*.

Bashir, I. (2017). Mastering Blockchain: Deeper insights into decentralization, cryptography, Bitcoin, and popular Blockchain frameworks. In *Packt Publishing*.

Brazil, B. (2018). *Prometheus : up & running : infrastructure and application performance monitoring*. https://books.google.com/books?id=QW1jDwAAQBAJ

Cachin, C. (2016). Architecture of the Hyperledger Blockchain Fabric. *Workshop on Distributed Cryptocurrencies and Consensus Ledgers (DCCL 2016)*.

Crosby, M., Nachiappan, Pattanayak, P., Verma, S., & Kalyanaraman, V. (2016). Blockchain Technology - BEYOND BITCOIN. *Berkley Engineering*. https://doi.org/10.1515/9783110488951

Dahmen, G., & Liermann, V. (2019). Hyperledger Composer—Syndicated Loans. In *The Impact of Digital Transformation and FinTech on the Finance Professional*. https://doi.org/10.1007/978-3-030-23719-6_4

Dinh, T. T. A., Liu, R., Zhang, M., Chen, G., Ooi, B. C., & Wang, J. (2018). Untangling Blockchain: A Data Processing View of Blockchain Systems. *IEEE Transactions on Knowledge and Data Engineering*. https://doi.org/10.1109/TKDE.2017.2781227

Fichman, R. G., Kohli, R., & Krishnan, R. (2011). The role of information systems in healthcare: Current research and future trends. *Information Systems Research*. https://doi.org/10.1287/isre.1110.0382

Getting Started - Covalent Documentation. (n.d.). Retrieved January 19, 2020, from https://docs.covalentx.com/article/71-getting-started

Guimarães, T., Silva, H., Peixoto, H., & Santos, M. (2020). Modular Blockchain Implementation in Intensive Medicine. *Procedia Computer Science*, *170*, 1059–1064. https://doi.org/10.1016/j.procs.2020.03.073

Hammi, M. T., Hammi, B., Bellot, P., & Serhrouchni, A. (2018). Bubbles of Trust: A decentralized blockchain-based authentication system for IoT. *Computers and Security*. https://doi.org/10.1016/j.cose.2018.06.004

Hevner, A. R., & Chatterjee, S. (2010). Design Research in Information Systems: Theory and Practice. In *Springer*. https://doi.org/10.1007/978-1-4419-6108-2

Hyperledger Caliper Architecture. (n.d.). https://hyperledger.github.io/caliper/v0.3.2/architecture/

Kuo, T. T., Kim, H. E., & Ohno-Machado, L. (2017). Blockchain distributed ledger technologies for biomedical and health care applications. In *Journal of the American Medical Informatics Association*. https://doi.org/10.1093/jamia/ocx068

Lin, I. C., & Liao, T. C. (2017). A survey of blockchain security issues and challenges. *International Journal of Network Security*. https://doi.org/10.6633/IJNS.201709.19(5).01

Meyerson, J. (2014). *The go programming language*. IEEE Software. https://doi.org/10.1109/MS.2014.127

Microsoft. (2020). *Documentation for Visual Studio Code*. Visual Studio Code. https://code.visualstudio.com/docs

Mohanta, B. K., Jena, D., Panda, S. S., & Sobhanayak, S. (2019). Blockchain technology: A survey on applications and security privacy Challenges. *Internet of Things*, *8*, 100107. https://doi.org/10.1016/j.iot.2019.100107

Olleros, F. X., & Zhegu, M. (2016). Research handbooks on digital transformations. In *Research Handbooks on Digital Transformations*. https://doi.org/10.4337/9781784717766

OpenJS Foundation. (2020). *About Node.js*. OpenJS Foundation. https://nodejs.org/en/about/

Pavan, A. (n.d.-a). *GitHub - BasicNetwork-2.0*. Retrieved February 5, 2021, from https://github.com/adhavpavan/BasicNetwork-2.0

Pavan, A. (n.d.-b). *GitHub - ContainerisingBlockchainExplorer*. Retrieved February 5, 2021, from https://github.com/adhavpavan/ContainerisingBlockchainExplorer

Pavan, A. (n.d.-c). *GitHub - ContainerisingCaliperForFabricBenchmark*. Retrieved February 5, 2021, from https://github.com/adhavpavan/ContainerisingCaliperForFabricBenchmark

Peffers, K., Tuunanen, T., Rothenberger, M. A., & Chatterjee. (2008). Peffers et al. (2008) A Design Science Research Methodology for Information Systems Research. *Journal of Management Information Systems*.

Postman Inc. (2021). *Postman | The Collaboration Platform for API Development*. Postman.Com. https://www.postman.com/

Prometheus. (2019). *Overview | Prometheus*. Https://Prometheus.Io/Docs. https://prometheus.io/docs/introduction/overview/

Singhal, B., Dhameja, G., & Panda, P. S. (2018). *Beginning Blockchain – A Beginner's Guide to Building Blockchain Solutions*. https://doi.org/10.1007/978-1-4842-3444-0

The Linux Foundation. (2020a). *Hyperledger Caliper – Hyperledger*. https://www.hyperledger.org/use/caliper

The Linux Foundation. (2020b). *Hyperledger Explorer*. https://www.hyperledger.org/use/explorer

The Linux Foundation. (2020c). *Hyperledger Fabric - Hyperledger*. https://www.hyperledger.org/use/fabric

Victor, J. M. (2013). The EU general data protection regulation: Toward a property regime for protecting data privacy. In *Yale Law Journal*.

Viriyasitavat, W., & Hoonsopon, D. (2019). Blockchain characteristics and consensus in modern business processes. *Journal of Industrial Information Integration*. https://doi.org/10.1016/j.jii.2018.07.004

Wang, R., Ye, K., & Xu, C. Z. (2019). Performance Benchmarking and Optimization for Blockchain Systems: A Survey. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics): Vol. 11521 LNCS*. Springer International Publishing. https://doi.org/10.1007/978-3-030-23404-1_12

Zheng, Z., Xie, S., Dai, H., Chen, X., & Wang, H. (2017). An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends. *Proceedings - 2017 IEEE 6th International Congress on Big Data, BigData Congress 2017*. https://doi.org/10.1109/BigDataCongress.2017.85

Zheng, Z., Xie, S., Dai, H. N., Chen, X., & Wang, H. (2018). Blockchain challenges and opportunities: A survey. *International Journal of Web and Grid Services*. https://doi.org/10.1504/IJWGS.2018.095647