INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




Distributed Ledger for Insurance Information Sharing

CATARINA FIGUEIREDO FRANCO
Outubro de 2020

POLITÉCNICO
DO PORTO

INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO

Distributed Ledger for Insurance Information Sharing

Catarina Figueiredo Franco

Dissertation to obtain the master's degree in Informatics Engineering, Software Engineering Specialization Area

Academic Advisor: Isabel de Fátima Silva Azevedo

External Advisor: José Cerqueira

Porto, October, 2020

Dedicatory

To my parents, my boyfriend, and my sister.

Abstract

The global insurance market has been continually adapting to challenges and new opportunities. Nowadays, most insurances have centralized systems, which could be inefficient and less transparent. Moreover, such solutions can also create distrust and conflicts of interest between insurance companies and policyholders.

Other recent technologies can improve existing business models. Distributed ledger technology, in specific, can offer to the insurance sector transparency and efficiency at another level, without compromising the security and privacy that the sector requires.

Distributed ledger technology avoids the single point of failure problem. This protection increases the security level of the system since it is more demanding, or nearly impossible, to compromise the data in the ledger.

This document presents a comprehensive overview of different distributed ledger technologies, which lead to the decision of the one that best suits the business case. The prototype design and implementation are fully described, along with the tests and the evaluation of the results.

A solution was explored to create a decentralized data storage, for workers' insurance information sharing, using a distributed ledger technology. This decentralized approach diminishes the responsibilities of the central authority by removing the project dependence upon centralized computing resources for storing and processing.

Keywords: Distributed Ledger Technology, Decentralized, Insurance, Work Accidents.

Resumo

O mercado global de seguros está continuamente a adaptar-se a desafios e novas oportunidades. Nos dias de hoje, a maioria das seguradoras possui sistemas centralizados, o que os torna ineficientes e menos transparentes. Um sistema centralizado também cria desconfiança e conflitos de interesse entre seguradoras e segurados. Existem tecnologias disruptivas que podem melhorar os modelos de negócio existentes fornecendo novas maneiras de pensar. A tecnologia de registos distribuídos pode oferecer ao negócio dos seguros uma solução mais transparente e eficiente, sem comprometer a segurança e a privacidade que o sector exige.

O objetivo principal desta tese é a criação de um sistema de armazenamento de dados distribuído para a partilha de informação de seguros de acidente de trabalho usando uma tecnologia de registos distribuídos. Esta abordagem descentralizada diminui as responsabilidades da autoridade central, removendo a dependência do projeto de recursos de computação centralizados para armazenamento e processamento.

A tecnologia de registos distribuídos permite a criação de um sistema protegido contra o problema do único ponto de falha uma vez que não há uma única máquina na rede que mantenha todos os registos de dados. Esta proteção aumenta o nível de segurança no sistema, pois é quase impossível comprometer os dados integralmente.

Em primeiro lugar este documento apresenta uma visão geral das diferentes tecnologias de registos distribuídos que levou à decisão da tecnologia mais adequada para o caso de negócio. Esta dissertação também apresenta uma descrição da escolha da plataforma usada para a imlementação do caso de uso. Por último apresenta o design e a implementação da aplicação, assim como os testes a avaliação de resultados.

Palavras-chave: Tecnologia de registos distribuídos, Descentralização, Seguros, Acidente de trabalho.

Acknowledgement

This journey would not have been possible without the support of those who directly or indirectly contributed to the project. I am very grateful to all and would like to express my sincere thanks.

First, I want to thank my parents, my boyfriend and my sister, for all the unconditional support, motivation, and for always believing in me.

Second, I want to thank my master advisor, Isabel Azevedo, for the availability, advises and dedication. It was truly amazing to have an advisor with such care and attention.

I also want to thank my advisors from Critical Software, Luís Figueiredo and José Cerqueira, who were always available and with great suggestions and improvements.

Lastly, I want to thank all my friends for the support and for helping me distract and rest my mind for some moments.

Table of content

List of figures

List of tables

List of code snippets

Acronyms & Definitions

AAIS	American Association of Insurance Services
AFAS	Applications for Administrative Solutions
AI	Artificial Intelligence
API	Application Programming Interface
APS	Portuguese Association of Insurers
BFT	Byzantine Fault Tolerance
CorDapp	Corda Application
DAG	Direct Acyclic Graph
DL	Distributed Ledger
DLT	Distributed Ledger Technology
DPoS	Delegated Proof of Stake
DSRM	Design Science Research Methodology
DSS	Decision Support System
FEI	Front End of Innovation
FFE	Fuzzy Front End
FinTech	Financial Technology
FTP	File Transfer Protocol
GDPR	General Data Protection Regulation
HTTP	Hypertext Transfer Protocol
IEC	International Electrotechnical Commission
InsurTech	Insurance Technology

IoT	Internet of Things
IP	Internet Protocol
IS	Information Systems
ISO	International Organization for Standardization
JDBC	Java Database Connectivity
JVM	Java Virtual Machine
MCDM	Multi-Criteria Decision-Making
ML	Machine Learning
NCD	New Concept Development
NPD	New Product Development
PBFT	Practical Byzantine Fault Tolerance
PoC	Proof of Concept
PoD	Proof of Deposit
PoET	Proof of Elapsed Time
PoS	Proof of Stake
PoW	Proof of Work
Premiums	Amount of money an individual or business pays for an insurance policy
P2P	Peer to Peer
P&C	Property and Casualty
REST	Representational State Transfer
RPC	Remote Procedure Call
SMTP	Simple Mail Transfer Protocol

SQL	Structured Query Language
SSH	Secure Socket Shell
TCP	Transmission Control Protocol
UML	Unified Modeling Language
XoL	Excess of Loss
WEF	World Economic Forum

1 Introduction

1.1 Context

In insurance, the policy represents a contract between an individual or entity and the insurance company. The finality of this contract is to receive financial protection against losses.

Insurance against accidents at work guarantees medical, hospital care and indemnification necessary to compensate for the damage suffered by a worker in the event of an accident occurring during working hours, or on the way to and from the workplace (Tocha, 2018).

In Portugal, all employers, whether singular or collective, are obliged to repair the consequences of accidents at work suffered by their employees. This legislation was forced in 1913 and is applied to every employer regardless of the size and employees number (Tocha, 2018).

1.2 Problem

The insurance market provides to consumers or groups many solutions that offer coverage for specific risks (Beers, 2015). Many of the organizations that take part in this mature market context share information about their insurance policies, including coverage, historical events, and claims. Currently, insurance data, confined in a centralized ledger - relational database - contains sets of private and sensitive information. The emerging concept of Distributed Ledger Technologies (DLT) introduces the ability to create more reliable, secure and trusted platforms, to store and access the information mentioned above taking advantage of the many properties related to these types of technologies. Being distributed one of the core properties of a

Distributed Ledger, it is possible, by relying on a pre-determined protocol, to share information between the entities that are part of the ecosystem (Anwar, 2018).

Insurance depends on sharing information between multiple parties to create quotes, underwrite risks, settle claims, or simply manage financial transactions. With DLT, the internal processes became more efficient - Data reconciliation is made more accessible, accuracy is improved, and time spent uncovering information eliminated, allowing for transparency, efficiency gains and cost reductions throughout a value chain (Lehman, 2019).

The problem of accidents at work has multiple impacts, affects several entities, and their consequences extend over time. However, the occurrence of accidents at work is not a fatality, as there is the possibility to intervene and minimise the respective consequences (Gaia, 2005).

A 2003 study about accidents at work in Portugal highlighted that (Gaia, 2005):

- Despite the downward trend in 2001, the occurrence of accidents had an 11% reduction resulting in 233,286 accidents in a context in which the employed population only presented a decrease of 0.67%.
- 5.6 million days of work lost, because of absenteeism caused by disabilities resulting from accidents at work, although also with a downward trend compared to previous years.
- One hundred eighty-one accidents occur in the workplace, with fatalities as a result. The number of fatal accidents between 1999 and 2003, decreased by 64.6%.
- The sector with the highest incidence of accidents at work is the manufacturing industry, with a higher rate of fatal accidents in the construction sector.

This collection of information on work accidents arose from the need to portray the situation of occupational accidents in Portugal since the knowledge of this reality allows the adoption of national prevention measures and periodic control of the results obtained, thus enabling a constant improvement health and safety conditions at work (Planeamento, 2013).

Accidents at work decisively affect the quality of life and are reflected in the national economy. These accidents are a concern that still exists nowadays and with distributed ledger technology can be a great benefit providing a solution where data can be collected in an up to date basis, and therefore, be used to identify new strategies to intervene.

Nowadays, the accidents at work's compensation claims process are managed by Portuguese Insurance Association (APS) that provides a software system for the hospitals and insurers to share information about the accident and reach a verdict. This system is centralized, which forces APS to provide the necessary characteristics – security, immutability, and transparency – and, consequently, have more responsibility being the owner of the platform and all the data.

1.3 Objectives

The main objective of this dissertation is to develop a prototype using distributed ledger technology that shares data related to Workers Compensation Insurance among insurance companies and hospitals on a specific protocol.

This project consists of three stages:

1. The first stage consists of a meeting with the representative of the Portuguese insurance companies to gather the requirements information.
2. The second stage consists of thorough research concerning the existing platforms based on Distributed Ledger Technologies (Blockchain, Directed Acyclic Graph, Hashgraph) to understand which of these options is more reliable to address the gathered requirements of the protocol. Furthermore, these two stages are crucial to understanding which technology is more suitable to the pre-defined protocol and, choose which platform better fits the needs related to this business context.
3. The third and last stage of this project relates to the prototyping of the work accident insurance system, considering the results obtained throughout the previous steps.

This dissertation, apart from the research intrinsic to the distributed ledger solution on the technical side, considers the complex business rules related to the insurance market to analyze and understand the pre-determined protocol that describes underlying practices for information sharing among entities. The contract also specifies which parties will take an active part in the prototyped system as well as the consensus mechanism (another core property of a DL).

The business use-case used for the development of this dissertation prototype belongs to the Portuguese Insurance Association and refers to the data management of worker's compensation claims. This business case is already implemented using a centralized system, and this

dissertation aims to verify if a distributed ledger technology can be beneficial to this type of business cases, reducing the responsibilities of the central authority, by distributing them among the participants on the network, and still providing security and transparency. The data ownership is shared across the participating peers instead of being centralized in APS, reducing attacks since there is not only one single point of failure.

The goal of this project is to generate value to the insurance sector by using the existing properties of a DLT and develop a proof-of-concept application that provides transactional improvements, efficiency improvements and data quality improvements generating a secure and private system.

1.4 Research Methodology

Design Science Research Methodology (DSRM) is a well-known and accepted framework to implement design science research in Information Systems (IS) research. This methodology will be used on the development of this dissertation's work and is composed of six steps (Peffers, Tuunanen, Rothenberger, & Chatterjee, 2007):

1. Problem identification and motivation: definition of the problem and justification of the value of a solution. Accepting the results after a reasonable understanding of the problem. This dissertation problem is described in section 1.2, and the value analysis is shown in the first chapter and detailed in chapter 1.
2. Definition of the objectives which can be seen in section 1.3. Knowledge gathering of the state of the problem and other solutions that can be seen in section 2.3 and 3.3.
3. Design and development of the prototype after gathered all requirements. The design and implementation of the solution are described in chapter 4 and chapter 4.6, respectively.
4. Demonstration: prove that the artefact can solve the problem or parts of it. It is usually accomplished through experimentation, simulation, case study or similar activities, that in this project were performed in chapter 5.
5. Efficiency evaluation comparing the objectives to the observed results. The evaluation of this document's solution is presented in chapter 5, section 5.1.1.
6. Communication with other researchers and audiences about the problem, the artefact, the design, and its results demonstrated. In this dissertation context, the communication phase is the presentation of the work to an evaluation committee.

1.5 Document Structure

This document is structured as follows:

- **Introduction.** The initial chapter covers this dissertation's context, along with the description of the problem and its main objectives, followed by the adopted methodology for the research and implementation stages.
- **Distributed Ledger Technology.** This chapter is divided into three parts. The first one provides a theoretical explanation of what a distributed ledger technology is along with an overview of its existing types, benefits, and obstacles. The second parts cover the explanation of the selected distributed ledger technology. The last part of this chapter describes how distributed ledger technology can be used for insurance information sharing.
-

- **Distributed Ledger** Technologies Platforms**.** This chapter describes the methodology used to evaluate which DLT platform best suits the business problem. Additionally, this chapter describes the two best platforms and presents the application that is currently using either one.
- **Solution.** This chapter begins to present the software requirements specification that leads to the choice of the platform being used for this PoC application. It also presents the domain modelling of the application, along with a platform overview. Additionally, this chapter shows the architectural design, and lastly, the implementation of the worker compensation insurance application.
- **Tests and Solution** Validation**.** This chapter dedicates to solution validation. It presents the implementation of different types of tests and the evaluation of the solution to conclude if it met the expectations.
- Finally, is the **Conclusions** chapter that describes the work summary, the contributions and the next steps.

Besides these chapters, some annexes were also included in this document, namely:

- **Attachment A.** Describes the promising areas of opportunity in the insurance value chain.
- **Attachment B.** Presents the innovation process of the application.
- **Attachment C.** Presents the decision structure used to choose the distributed ledger technology platform.
- **Attachment D.** Includes the webserver postman collection used to perform the functional tests.

1.6 Insurance Context

The insurance market provides consumers or groups with many solutions that offer coverage for specific risks. Many of the organizations that take part in this mature market context share information about its Insurance Policies, including coverage, historical events, and claims.

The Portuguese Association of Insurers (APS) is a business association that represents the insurance companies operating in the Portuguese market whose leading roles include:

represent and safeguard the interests of insurance companies at national and international level; promote cooperation between associates and, organise and manage services of common interest to all associates (Portugal, 2018).

Decisions related to insurance coverage need to be made in brief periods rather than days or even weeks. In an era where retail products can be ordered and delivered on the same day, the insurance market should aim to a faster, more convenient, and secure service provider.

The term FinTech has described as being "technologically enabled financial innovation that could result in new business models, applications, processes, or products with an associated material effect on financial markets and institutions and the provision of financial services" (Board, 2019).

InsurTech is the insurance-specific branch of FinTech that refers to the variety of emerging technologies and innovative business models that have the potential to transform the insurance sector. The main types of innovations that fall within the scope of InsurTech include (Supervisors, 2017):

- Digital platforms (internet, smartphones).
- Internet of Things (IoT).
- Telematics / Telemetry.
- Big Data and Data Analytics.
- Comparators and Robo advisors.
- Machine Learning (ML) and Artificial Intelligence (AI).
- Distributed Ledger Technology (DLT), Blockchain and Smart Contracts.
- Peer-to-peer, Usage-Based and On-Demand Insurance.

Currently, APS has a centralized solution to process worker's compensation claims. This solution consists of a client-server application, used by the hospitals and insurance companies, to follow a claim process.

This process consists of three phases:

1. **The hospital reports the injury to the insurer**

When a work accident happens, the employee goes directly to the hospital to receive treatment. When the employee is admitted, the hospital initiates the work claim process, introducing the initial data related to the patient and his policy. Once the process has all the necessary data, including the required medical documents proving the injury, the process is submitted and sent to the insurance company.

2. **The insurer updates the process**

   When the insurer receives the workers' compensation claim process, it can choose to accept the request or to decline. If the process is recognized, the insurer fills all the necessary information about employee insurance. In both cases, a message is sent to the hospital informing about the result of the claim.

3. **The hospital receives the insurer response**

   For a positive response, the hospital creates an invoice with all medical acts performed to the employee. The hospital submits the invoice, and the insurer receives a message with the invoice information.

   In case of a negative response, the hospital closes the process.

The business process model diagram for a worker’s compensation claim is described in Figure 1.

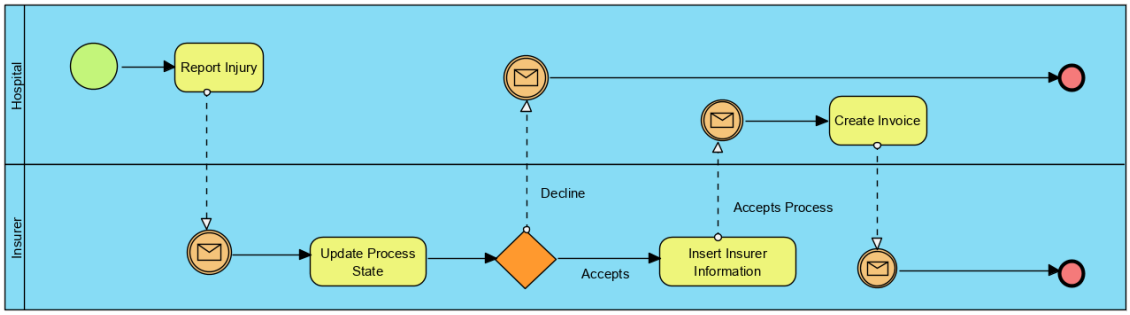


Figure 1. The process model for Worker's compensation claim

2 Distributed Ledger Technology

As previously stated, a distributed ledger is a database that is updated and held by every member independently in an ample network space. In this type of ledger, there is no central authority that process and stores the data. Instead, all nodes will hold the ledger and construct it independently. Nevertheless, in that case, the nodes on the network will need to have access to the transaction list and give out their conclusion before adding it on the distributed ledger (Anwar, 2019).

Usually, every node on the network goes through an agreement process to reach a single conclusion. This process differs from distributed ledger to distributed ledger.

2.1 Technology Context

This section presents the different types of distributed ledger technology, how they work, their principles, benefits, and challenges.

The three main distinct concepts of distributed ledger technology were highlighted by Daniel Burkhardt and Nabil El Ioini in (Burkhardt, 2018) and (El Ioini, 2018) respectively.

2.1.1 Blockchain

In 2008, *Satoshi Nakamoto wrote* a paper entitled *Bitcoin: Peer-to-Peer Electronic Cash System* about peer-to-peer electronic cash (Nakamoto, 2008). It introduced the term chain of blocks. This term evolved over the years into the word blockchain.

Blockchain is a distributed, decentralized, and immutable ledger that stores information in the form of blocks. The data is append-only and can be accessed by all the network participants (El Ioini, 2018).

The key characteristics of blockchain are (Z., 2017):

- **Decentralization**. In contrast to a centralized system, in a blockchain, there is no central authority, and all nodes talk to each other directly.
- **Persistency**. Transactions can only be added in a time-ordered sequence which means that one data is added, it is almost impossible to change the transaction data, making it practically immutable - any validated record is irreversible and cannot be changed.
- **Anonymity**. The blockchain participants can interact with each other with a generated address and without revealing their real identity.
- **Auditability**. In a blockchain system, transactions can be either unconsumed or a consuming state. The consumed transactions represent the immutable ledger state and can be used for audit proposes.

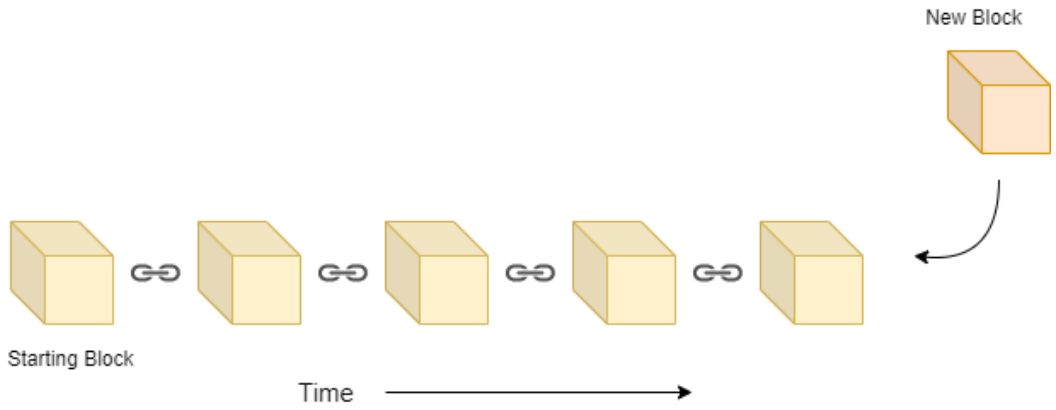


Figure 2. Blockchain illustration

In a typical blockchain, a block contains a group of transaction bundled together and sequencing organized. A transaction represents an event; for example, the event of transferring cash from a sender's account to a beneficiary's account. The block size varies depending on the type and design of the blockchain in use.

Every block will contain a unique ID known as the hash to differentiate or synchronise the transactions. This hash function helps to distinguish between all the transaction blocks on the

ledger. Mainly the function includes alphanumeric characters, and every hash function is a unique and random selection.

Each block in a blockchain contains a cryptography hash of the previous block, “chaining” the blocks to each other (Figure 3). Therefore, historical transactions in the blockchain may not be deleted or altered without invalidating the chain of hashes (Xu et al., 2017).

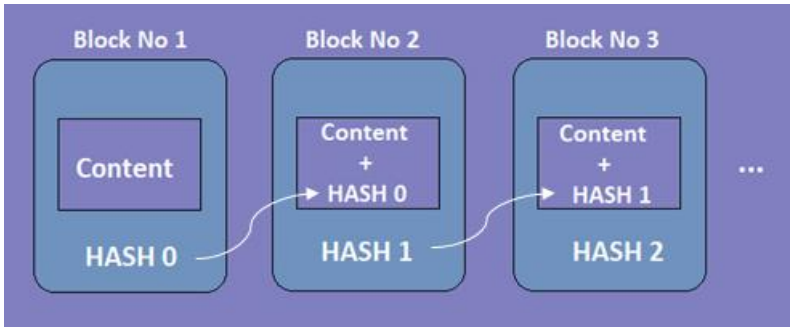


Figure 3. Blockchain Blocks

Publishing a whitepaper in 2008 and launching the initial code in 2009, Satoshi Nakamoto created the bitcoin to enable the peer-to-peer transfer of money without the need for a central authority, like a bank, to control and operate the ledger (Wales, 2019).

Other blockchains include those that run the several hundred “altcoins” – other similar currency projects with different rules – as well as genuinely various applications, such as (Wales, 2019):

- Ethereum is the second-largest blockchain implementation after bitcoin. Ethereum distributes a currency called ether, but also allows for the storage and operation of computer code, allowing for smart contracts. The system went live in July 2015.
- Ripple is a public ledger that provides real-time payment settlements and currency exchange services.

The increasing focus of global companies towards investment to develop and deploy blockchain projects is a crucial factor contributing to the growth of the worldwide blockchain technology market.

Bitcoin—the first and most infamous application of blockchain - market value surged from less than $20 billion to more than $200 billion throughout 2017 (Carson, 2018).

The blockchain market is still growing and, despite the numerous projects that are created every year, the recipe for success has not yet emerged.

The development of blockchain solutions without the proper evaluation of its value or feasibility can compromise the experimentation and lead to no return on the investments (Carson, 2018).

To prevent this from happening, the *McKinsey* company performed an evaluation and stress test to the impact and feasibility of blockchain opportunities grouped by the industrial sector. Figure 4 shows the analysis results.

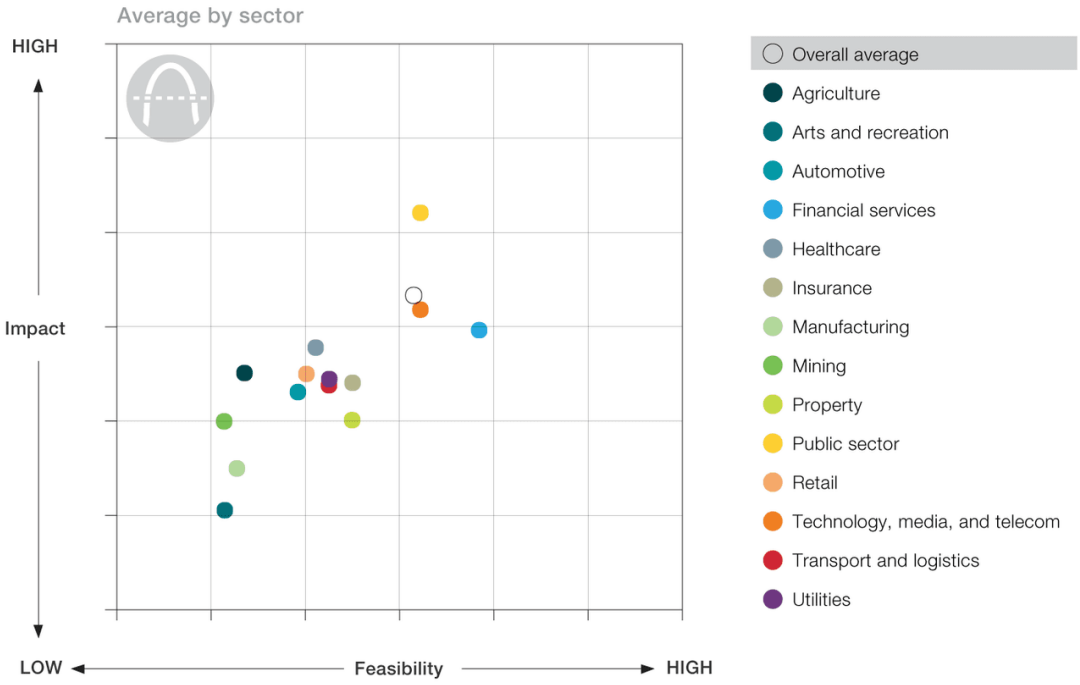


Figure 4. Blockchain opportunities by the industrial sector

Source: (Carson, 2018)

From the performed study, it is possible to conclude that the insurance sector has middle impact and intermediate feasibility for a blockchain implementation opportunity, which leads to a returning in the investment. Compared to the overall average, the insurance sector is still a little behind in terms of feasibility and impact. Compared to the other industries, it is well rated, having ahead of the Healthcare, Financial Services, Technology and Public sector, which are the sectors that have a higher impact on business opportunities.

2.1.2 Tangle

The main difference of a tangle ledger to a blockchain ledger is the data structure. In tangle, data is structured as a Directed Acyclic Graph (DAG) where the nodes represent transactions, and the edges indicate the direction between two transactions (Burkhardt, 2018).

A DAG is comprised of three fundamentals concepts:

- A **Graph** is composed of a series of vertexes connected by edges.
- Being **Directed** means that the edges are connected so that each side only goes one way.
- **Acyclic** means that it is impossible to start at one point in the graph and traverse it entirely. Each edge, in the graph, is directed from an earlier edge to a later edge.

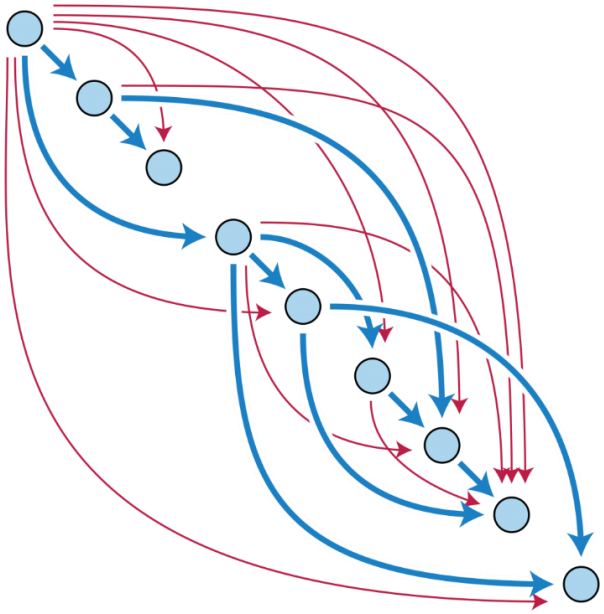


Figure 5. Directed Acyclic Graph illustration

Source: https://commons.wikimedia.org/wiki/File:Transitive_Closure.svg

Figure 5 shows a representation of a DAG painting in blue. The red edges connect the pairs of reachable vertices throughout the DAG chain.

To change a state it is necessary to have a input transaction. The tangle ledger store transactions in nodes where each node holds a single transaction.

Any node can initiate transactions; however, to validate them, they have to verify at least two of the previous transactions on the ledger. The more a node validates, the more its transactions become valid on the distributed ledger database. So, if a transaction has a long branch of previously approved transactions, it will carry the most weight in the ledger. However, an algorithm will randomly select the previous two transactions for each member to validate (Anwar, 2019).

This validation process is a new form of consensus to achieve greater scalability and holds a tremendous potential providing platform for speedy transactions.

Launched in November 2013, NXT was the first platform to utilise tangle (Jelurida, 2013). The other two noticeable implementations are:

- IOTA Tangle. Created in late 2015, IOTA Tangle is a cryptocurrency platform for the Internet-of-Things (IoT) industry (Popov, Saa, & Finardi, 2019).
- ByteBall. Launched on Dec 25, 2016 (Obyte, 2016), Byteball is a decentralized system that allows the storage of data, including transferrable assets such as currencies, property titles, debt, etc. (Churyumov, 2016).

2.1.3 Hashgraph

Hashgraph is a distributed data structure that, like the others, maintain their data among a large number of network peers. Hashgraph, released in July 2017, is a new alternative to blockchain technology. The most significant innovation is the usage of a gossip protocol where each node in Hashgraph can spread signed information (called events), on newly created transactions and transactions received from others to its randomly chosen neighbours. These neighbours will aggregate received events obtained from other nodes and send it to another randomly chosen neighbour. This process, called gossip sync, is finished when all nodes are aware of the information created or received at the beginning (Hoxha, 2018). When this happens, each participant sores the event in a memory data structure that contains a timestamp, an array of transactions. The parents' hashes are the hash of the last event created before the gossip sync and, the other parent hash is the hash of the previous event produced before the gossip sync (Hashgraph, 2019).

The event's history can be represented by a graph (Figure 6) where each member is one column of vertices (Baird, 2016).

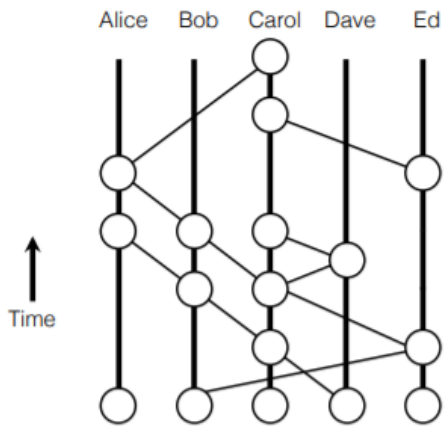


Figure 6. Hashgraph illustration

Source: (Baird, 2016)

The vertices in each column represent a gossip event. In the Alice column, for example, the top event represents Bob performing gossip sync to Alice, in which Bob sent her all of the information that he knew (Baird, 2016).

In a typical gossip protocol, a diagram like this is merely used to discuss the protocol; there is no actual graph stored in memory anywhere. Each event is stored in-memory data structure as a sequence of bytes and signed by its creator (Baird, 2016).

Further, by performing virtual voting, each node can determine if a transaction is valid based on whether it has over two-thirds of nodes in the network witnesses.

Hashgraph was developed by Leemon Baird in 2016 and was initially intended for the private corporate sector (Hays, 2018).

With the project success in the corporate sector, Hedera launch “Hedera Hashgraph Platform”, which aim to drive forward the patented Hashgraph technology for the development of a public Hashgraph network. The source code might be publicly available to anyone that wants to become part of the Hedera Hashgraph community, as a network node, but the project still has a governance model and the leadership council that is composed of thirty-nine organizations. This limitation forbids the fork of the source to create an alternative project (Hays, 2018).

2.1.4 Summary

From the previous explanation, it is apparent that all DLTs share common aspects of transparency, consensus, transactional, distributed, peer-to-peer and flexible. However, differential aspects emerge in the consensus mechanism used to approve the transactions, and in the structure of data within each DLT.

This section presents a summary of the DLT mentioned above. The main aim of this section is to decide which technology better suits this dissertations business context – insurance against accidents at work.

The comparison based on the functional data structures is for maintaining the ledger, transaction validation, ledger size, scalability, and popularity.

For the context of this dissertation, an essential requirement that the DLT should have is security. Insurance data is susceptible, which requires the minimization of risks by using mature and accessible distributed ledger technology. Data must be resistant to attacks even with low transactions volume.

Table 1. Distributed Ledger Technologies Summary

	Hashgraph	**Tangle**	**Blockchain**
Data Structure	Data is stored in a graph-like structure	Tangle structure stores its transactions in nodes, where each node holds a single transaction	Data is structured in blocks in the form of transactions that are validated by miners in the ecosystem
Validation of transactions	The gossip protocol and virtual voting ensure that the majority validates transactions.	All members in the network verify transactions which represent an increase in the number of members, resulting in fast validation.	Periodically batch up transactions needing confirmation — into a block — and confirm them in one go using a consensus mechanism

Transactions per second	The unique consensus protocol reduces the computational load and provides higher scalability and higher TPS	Using a DAG data structure ensures high scalability and high TPS	Limited scalability and TPS
Patented	Yes	No	No
Limitations	The technology has only been deployed in a private network although, the real potential will only be known once it is released in a public network (Takyar, 2018) Any new invite to the ledger will rely on and go through Swirlds	Low transactions volume makes tangle more vulnerable to attacks (Hays, 2018)	Low transaction speed High power consumption

In terms of transactions volume, a DAG implementation when faced with low transactions volume can be sceptical to attacks which can lead to security breaches resulting in the ledges instability. Since this project will not deal with a massive amount of data (big data) neither with high operational demand, the complexity of a DAG implementation does not justify the business needs.

Since insurance data is composed of private and sensitive information, the speed and transaction fees become less critical, while security becomes the primary concern.

In conclusion, despite its inefficiency, blockchain technology continues to be the most tested, used, and versatile DLT available today, and is the chosen DLT for this dissertation proof of concept.

2.2 Blockchain Overview

Blockchain can be imagined as a layer of a distributed peer-to-peer network running on top of the internet, as Figure 7 describes. The blockchain is analogous to SMTP, HTTP, or FTP running on top of TCP/IP (Bashir, 2018).

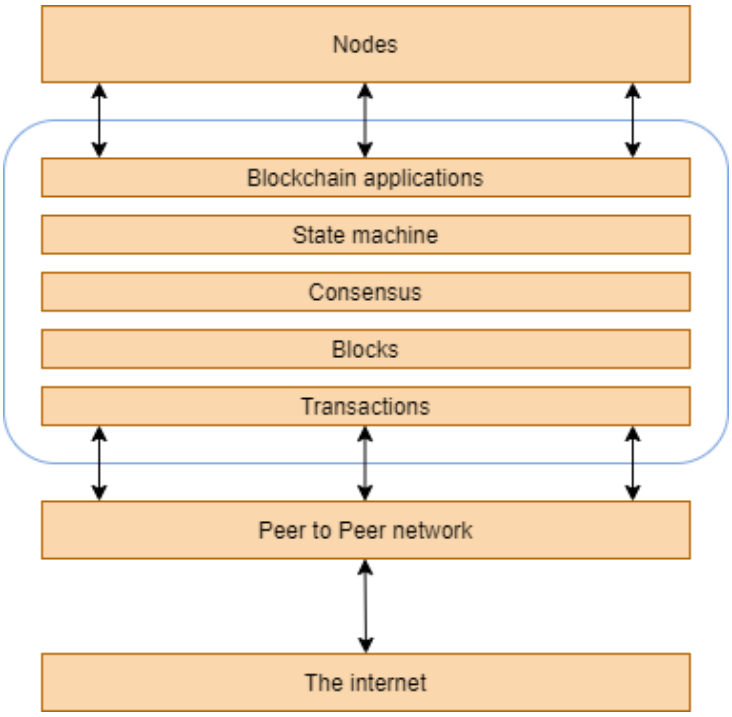


Figure 7. Overview of a blockchain network

At the bottom layer, in Figure 7, there is the internet, which provides an underlying communication layer for the network. On top of the internet is the peer-to-peer network, that contains transactions, blocks, consensus mechanisms, state machines, and blockchain smart contracts. Finally, at the top, some users or nodes connect to the blockchain and perform various operations such as consensus, transaction verification, and processing. Nodes also perform a transaction signing function. Nodes create transactions that are later digitally signed by the legitimate owner of the asset, using private keys as proof of ownership (Bashir, 2018). This process is shown in Figure 8, where three transactions are presented, and each transaction is digitally signed with the owner private key.

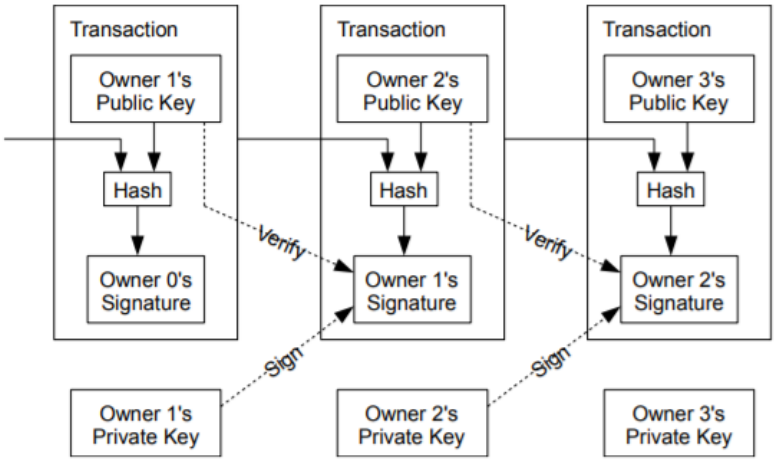


Figure 8. Blockchain Transactions structure

Source: (Nakamoto, 2008)

Blockchain technology can be implemented in various economic sectors, particularly in the finance sector, where improvements in the performance of financial transactions can be significantly beneficial (Bashir, 2018).

Figure 9 shows a broad-spectrum outline of year-wise progression and adaption trend of blockchain technology. Years shown on the x-axis indicate the range of time in which a specific phase of blockchain technology falls. The y-axis shows the level of activity, involvement and adoption of blockchain technology (Bashir, 2018).

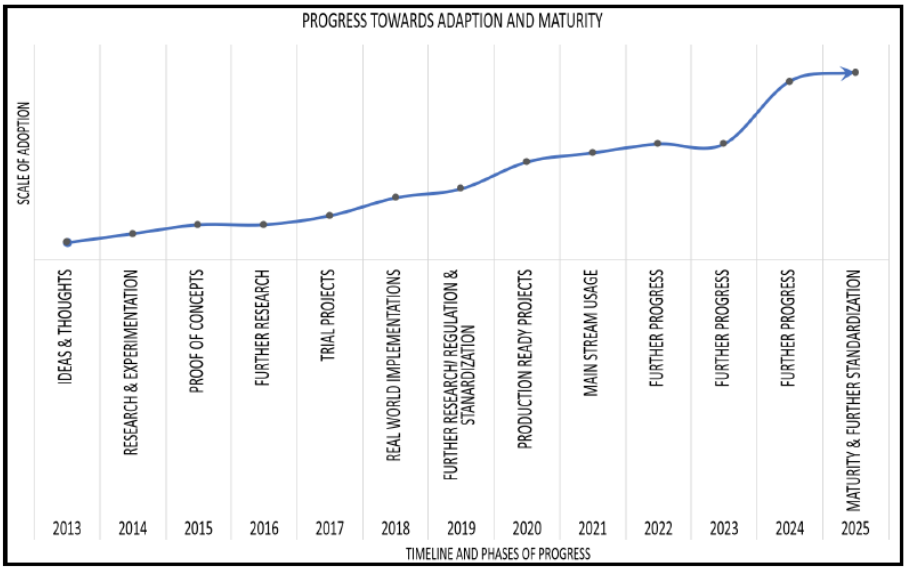


Figure 9. Blockchain technology adoption and maturity

Source: (Bashir, 2018)

Figure 9 graph is based on researches made in recent years, that suggest that by 2025 blockchain technology is expected to become mature (Bashir, 2018).

The graph also shows that in 2013 appeared the first ideas of the usage of the blockchain outside the cryptocurrency area. The cryptocurrencies area was already the primary usage of blockchain technology and many new coins emerged during that time (Bashir, 2018).

2.2.1 Categories of Blockchain

Currently, blockchain systems are categorised into three types (Z., 2017):

- **Public.** This type of blockchain is open to everyone interested in participating and become a node in the blockchain. All network users share the records; each user maintains a copy of the ledger and uses the consensus mechanism to decide the state of it. Bitcoin and Ethereum are both public blockchains.
- **Private**. In this case, the blockchain is private, which means that only a group of individuals or organizations have access to the ledger and can share the ledger among themselves.

  Hyperledger and R3 Corda are an example of private blockchains.
- **Consortium**. In a consortium blockchain, only a group of pre-selected nodes can participate in the consensus process. Currently, Hyperledger is developing a business consortium blockchain framework (Z., 2017).

2.2.2 Consensus

The consensus is the process that ensures that all nodes, in a distributed system, are consistent with each other.

In a blockchain, there is no central authority that ensures that different nodes have the same ledger and are consistent. Therefore, it is necessary to implement a consensus approach. This section presents some of the most known consensus mechanisms used in the blockchain environment (Z., 2017).

All the consensus mechanisms fall into two main categories:

- Proof-based or leader-election lottery-based consensus is a type of agreement where a leader is elected randomly (using an algorithm) and proposes the final value. This type of category refers to the public types of consensus mechanisms. Some of the above mention consensus mechanisms that fit into this category are, for example, Proof of Work, Proof of Stake and Proof of Elapsed time.
- BFT-based, on the other hand, is a traditional approach based on rounds of votes. This type of consensus requires that every node be known to the network (Z., 2017). Example of this type of category is, for instance, PBFT.

Proof of Work (PoW)

In PoW, only one node can be selected to record a block. This selection cannot be random because it would make the system vulnerable to attacks. Therefore, to select a trusty node is necessary for some computer calculations. In PoW, nodes compete with each other to search for a nonce, by sheer brute-force use of processing power, until the resulting block hash follows a particular pattern of a certain number of zeros (Zheng, Xie, Dai, Chen, & Wang, 2017). In this consensus, the selection of the node is purely based on the proportion of their computing capacity.

When one of the nodes reach the target value, it broadcasts the block to other nodes that must confirm the correctness of the hash value. If the block is valid, the miners (nodes that calculate the hash values) will append this new block to their blockchains.

This consensus mechanism is used by Bitcoin (Nakamoto, 2008), and is designed to specifically address Byzantine faults (Lai & Lee Kuo Chuen, 2018).

Proof of Stake (PoS)

Proof of Stake was first introduced by Peercoin, in 2012 (Peercoin, 2012) and is an energy-saving alternative to PoW.

The selection process, in this consensus, takes into consideration the proportion of tokens a node already owns. It is believed that the more tokens a node possess, the less are the probabilities of this node attacking the network (Zheng et al., 2017). To lunch a successful attack, the node must acquire many tokens which makes the attack economically unsustainable (Lai & Lee Kuo Chuen, 2018).

Centralizing the power of the network in the richest nodes makes PoS more effective than PoW but, since the mining cost of PoS is low (practically zero), the system is more likely to suffer an attack. Many blockchains are adopting, at first PoW, and then gradually move to a PoS (Zheng et al., 2017).

Delegated Proof of Stake (DPoS)

This consensus approach is similar to PoS but with one difference – the nodes that have the more significant stake can delegate the generation and validation of a transition to other nodes by voting.

BitShares blockchain is currently using this consensus (BitShares, 2014).

Proof of Elapsed Time (PoET)

This consensus mechanism was introduced in the Hyperledger Sawtooth Lake project proposed by Intel in 2016 and attempts to address the random leader election problem (Holbrook, 2020). In PoET, each node requests a signed timer from a trusted function and the node that has the shortest waiting time, for a particular transaction block, becomes the leader and commits the next block to the blockchain (Lake, 2016).

By randomly distributing the leadership election across the entire network, this consensus mechanism tries to mitigate any potential malicious node that attempts to break the system by consistently receiving a shorter timer and consequently commit more blocks.

Proof of Deposit (PoD)

In this consensus mechanism, the nodes that wish to participate in the network must make a security deposit to mine and propose blocks. Tendermind blockchain is currently using this consensus (Tendermint, 2014).

Paxos

Lamport proposed Fault-tolerant consensus protocol in his seminal paper (Lamport, 1998). This consensus protocol is tough to implement as it is designed to handle all cases of non-Byzantine

faults (Lai & Lee Kuo Chuen, 2018). Paxos guarantees that consensus is achieved even in the presence of faulty nodes. It also ensures that the values will eventually reach all nodes.

Google's Chubby lock service is currently using Paxos (Burrows, 2006).

Raft

Raft is another fault-tolerant consensus protocol created by Diego Ongaro and John Ousterhout in 2014 (Ongaro & Ousterhout, 2014). Raft was designed to be easier to understand and implement, making this consensus protocol simpler than Paxos.

Raft has had a massive adoption, especially in the open-source community. Apache Kudu is currently using Raft (Kudu, 2019).

Both protocols are leader election-based consensus mechanisms, which means that nodes are competing in a leader-election lottery, and the node that wins proposes a final value (Bashir, 2018). The critical difference in the afore mention protocols is that Raft selects the leaders among the new nodes while Paxos select the leader among all nodes (Lai & Lee Kuo Chuen, 2018).

Practical Byzantine Fault Tolerance

In 1999, Castro and Liskov published **Practical Byzantine Fault Tolerance** or PBFT, that provided a BFT for practical use (Castro & Liskov, 1999). It describes the first state-machine replication protocol that correctly endures Byzantine faults in asynchronous networks. Their algorithm "can be used to implement any deterministic replicated service with a state and some operations" (Castro & Liskov, 1999).

PBFT guarantees liveness – progress will be made within the system – and safety – the chance that something negative will happen in the system – being able to handle up to 33% of nodes being faulty (Lai & Lee Kuo Chuen, 2018).

2.2.3 Network

Nodes are either miner who creates new blocks and mint cryptocurrency (coins) or block signers who validate and sign the transaction.

In a typical blockchain network, nodes compete with each other to be able to append a block in the blockchain. The fastest node to achieve consensus will be the one that will append the block (Bashir, 2018).

According to Satoshi Nakamoto, a typical blockchain network begans to broadcast the transactions to all nodes, then, each node collects this new transaction into a block, and tries to find the proof-of-work for its block. When a node finds the PoW, it broadcast the block to all nodes in the chain. The nodes accept the block if all transaction in it are valid an not already double spent. The process of acceptance is known when a node uses the accepted hash block to use it as previous hash in the next block (Nakamoto, 2008).

The longest chain is always considered the correct one. Nodes find the longest chain and work to extending it. In the eventuality of receiving two different versions of the block simultaneously, nodes will work with the first one received but will save the other one in case it becomes longer (Nakamoto, 2008). The broadcasting of new transactions does not necessarily reach all nodes. However, as long as they reach many nodes, the transactions will be gathered into a block before long (Nakamoto, 2008).

2.3 Distributed Ledger Technology for insurance information sharing

This section introduces the value of distributed ledger technology. It is divided into two parts, where the first part presents how DLT is helping the insurance market, and the second part describes how can DLT add value to data sharing.

2.3.1 DLT for insurance

During the last ten years, technology has fostered transformation and innovations in every sector, carrying around exciting applications and cutting-edge business models (Nicoletti, 2017).

In the insurance sector, DLT promises to drive efficiency in business practices and mitigate certain existing risks - enabling more efficient claims handling by automating business processes - and - limiting the scope for disagreement between parties (Forum, 2019).

A distributed ledger is a database that is shared and synchronised across a network of multiple sites, geographies, or institutions. It offers the capacity to deliver a new kind of trust to a wide range of services (Adviser, 2016).

Parties will be able to maintain accurate, shared records of financial agreements without duplications, alterations, reconciliations, or failed matches. A distributed ledger provides transparency by sharing the information between nodes and not relying on a centralized party to store the data (Forum, 2019).

DLT creates and maintains a “single version of the truth" between multiple parties, thereby reducing time-consuming and cumbersome exchanges as well as the reconciliation of many documents. When a secure single source of truth is available, audits and regulatory inspection can also become much more reliable and efficient (Forum, 2019).

The most notable features of distributed ledgers (DLs) are resistance to censorship, decentralized maintenance, and elimination of the need for a trusted third party (Benčić & Žarko, 2018).

In the insurance context, a DLT enables more efficient claims handling by promoting trust and transparency between parties. The data records in a distributed ledger provide traceability and availability, which can be used to minimised fraud (Forum, 2019).

According to “The Future of Financial Infrastructure” report, a DLT offers six key value drivers (World, 2016):

1. Operational simplification. DLT reduces/eliminates manual efforts required to perform reconciliation and resolve disputes.
2. Regulatory efficiency improvement. DLT enables real-time monitoring of financial activity between regulators and regulated entities.
3. Counterparty risk reduction. DLT challenges the need to trust counterparties to fulfil obligations as agreements are codified and executed in a shared, stable environment.
4. Clearing and settlement time reduction. DLT disintermediates third parties that support transaction verification/validation and accelerates settlement.

5. Liquidity and capital improvement. DLT reduce locked-in capital and provides transparency into sourcing liquidity for assets.
6. Fraud minimization. DLT enables asset provenance and full transaction history guaranteeing a single source of truth.

DLT will transform the way information is transferred and verified in most areas of the insurance value chain focusing mainly on the creation of immutable insurance claim records, development of asset provenance to assist in risk profiling and claims processing and P2P insurance (Board, 2019). In attachment A, it is possible to see the most promising areas of opportunity in the insurance value chain. Furthermore, in attachment B is possible to see the innovation process of this worker compensation insurance application.

2.3.2 DLT for data sharing

From the data management perspective, a blockchain is a database built on a peer-to-peer network providing trusted data management capabilities. This database management system ensures the trustworthiness of the system at three levels (Lu, 2019):

- The reliability of storage.
- The credibility of the processing.
- The reliability of external access.

This type of trust is guaranteed through consistency, traceability, and availability that blockchain technologies can provide data management quality.

In a decentralized system, data is allocated to each node, instead of being accumulated to form an information resource centre. The blockchain technology implements a ledger that chains structure data in linking data (Lu, 2019).

The aggregation of information in the block can be categorised into two ways:

1) On-chain storage. Information is stored directly in the blockchain. This approach needs to be considered taking into consideration that all the stored information will be available to all parties in the system (Thomas, Matthew, Philip, Alexander, & Bela, 2018).
2) Off-chain storage. Information is stored outside of the blockchain, in a local server, such as the cloud. This approach is necessary when parties want to verify information with

the blockchain but do not want to make the information available to the other parties. The size of the block is also an essential aspect since the information stored may be larger than the blockchain itself supports (Thomas et al., 2018).

In a peer-to-peer network, the blockchain accumulates the information through linked blocks and constructs the sharing of information through the copy of the blockchain (Lu, 2019).

Besides the information accumulation, the blockchain also has the function of sharing these data. Therefore, the information sharing on a blockchain is consistent with the blockchain network construction and scale expansion. This expansion is used to achieve a more extensive sharing of information - the more devices and individuals join the blockchain network, more information can be shared (Lu, 2019).

Data management has the purpose of keeping data as confidential, reliable, and available as possible. As previously said, in a blockchain system, each node stores all the historical transaction data. This way, while it is possible to guarantee transparency and high availability, it is difficult to ensure privacy and performance. The trade-off between preserving privacy and enhance transparency is something that must be taken into consideration.

Healthcare has had much investment in this technology area, and there are already two applications to share information about medical data:

- MeDShare is a medical data-sharing system between cloud service providers that make use of a blockchain. In this system, cloud service providers will be able to securely achieve and audit data while sharing medical data among other cloud service providers and entities like research and medical institutions that will make use of the medical data without any risk on data privacy (Xia et al., 2017).
- MedBlock is a blockchain-based information management system that handles patient's information, allowing the access and retrieval of electronic medical records. Contrarily to MeDShare, MedBlock does not store its data on the cloud to ensure that other agencies cannot access the original medical data of patients (Fan, Wang, Ren, Li, & Yang, 2018).

Using a peer-to-peer decentralized data sharing system promotes sharing efficiency and reduces potential data-related costs, which makes this a cutting-edge technology in terms of transformation and innovation. Technology, although it is not the single driver of disruption. Customers have significantly changed, namely their expectations and needs. To adapt to these

new changes, insurance companies have to put the customer at the centre of their strategies, being active and not reactive, to changes and innovations.

The secure and convenient sharing of personal insurance data is crucial to improve the interaction and collaboration of the insurance industry. Faced with the potential privacy issues and vulnerabilities that exist in the current insurance data storage and sharing systems, the insurance sector heartily benefits from this new decentralized solution.

3 Distributed Ledger Technologies Platforms

Nowadays, blockchain technology has attracted much attention and the number of platforms that have emerged, to build a blockchain application, is rapidly increasing, which means that the selection of the best blockchain platform, for software-producing organizations, is becoming a significant challenge.

This selection process consists of choosing and evaluating the blockchain platform that best fits the preferences and requirements of the software product to be developed, taking into consideration its future needs.

Acquiring the necessary knowledge to decide the best platform is a time-consuming process that requires the accurate interpretation of the domain context and the comparison of every alternative. This decision-maker process gets more complicated as the number of alternatives increases.

In 2019, the Amuse research team from Utrecht University, in collaboration with the Free University of Amsterdam and the AFAS Software in the Netherlands, introduced a technology selection framework to assist decision-makers, at software-producing organizations, with the multicriteria decision-making (MCDM) problem by building decision models and find suitable alternatives based on their requirements and priorities (Farshidi, Jansen, España, & Verkleij, 2020).

This Decision Support System (DSS) takes into consideration which blockchain platforms are available at the moment, the capabilities of the blockchain systems, and which features are fulfilled by which blockchain platform (Farshidi et al., 2020).

Thirteen experts – three DSS experts, four blockchain researchers from Dutch research institutes, two blockchain developers, and four blockchain consultants/public speakers – participated in the research to evaluate de decision support system and the decision model for the blockchain platform selection problem (Farshidi et al., 2020).

Comparing to other studies that address the blockchain platform selection problem, this DSS is the complete one, considering 121 criteria and 28 blockchain platforms alternative to build the decision model for the blockchain platform problem (Farshidi et al., 2020).

These research study criteria are composed of two sets – Qualities and Features. Software qualities such as interoperability, maturity, and performance of blockchain platforms are kept in the collection of Qualities defined by the ISO/IEC 25010 standard and extended ISO/IEC 9126 standard. Blockchain features like smart-contracts and consensus mechanisms were primary identity by blockchain experts and secondly through documentation and literature study of blockchain platforms. The blockchain features in this DSS is a collection of Boolean and Non-Boolean functions, where Boolean mean that 1 – the blockchain platform supports the feature, and 0 – the blockchain platform does not support the functionality. On the other hand, the Non-Boolean features are not so straightforward and cannot have a value of 0 or 1. These elements are used to analyze blockchain features based on their impact on quality attributes of blockchain platforms (Farshidi et al., 2020).

Usually, a single optimal solution for an MCDM problem does not exist, and it is necessary to employ decision-makers preferences to differentiate between solutions. Therefore, the MCDM proposed in this research study provides a list of prioritised options for decision-makers based on the selected features (Farshidi et al., 2020).

Decision-makers prioritise the blockchain features requirements using MoSCoW prioritization technique to assign blockchain features weights. The MoSCoW technique defines four priority groups: “MUST have”, “SHOULD have”, “COULD have”, “WILL NOT have” where requirements assigned to these groups are according to the importance of having them implemented.

The DSS result list is defined according to the decision-makers blockchain features requirements and priorities. A feasible blockchain platform must support all blockchain feature requirements with must-have priorities.

The efficacy and effectiveness of this decision model were evaluated through three case studies at three different blockchain software development companies. The case-study participants

identified the potentially feasible blockchain platforms for their organizations through multiple internal expert meetings and investigations of the various blockchain platforms before participating in this DSS research (Farshidi et al., 2020). The idea is to compare this list of feasible blockchain platforms chosen by the companies, with the outcomes of the DSS. The results were successful, and the DSS recommended nearly the same solutions as the case-study suggested to their companies but with more efficiency.

For this dissertation context, here is the identified features and their priority according to the requirements:

- The intended system requires integration with the current system (e.g. event streaming), and, therefore, *enterprise system integration* is considered a must-have feature.
- Being a *private* and *permissioned* platform is a must-have feature since the number of users is limited, and the data must be private.
- The *protocol layer* supports reaching consensus on the accuracy of the data, incorporate rules about transactions validation and network participation and therefore, is considered a must-have feature.
- The *network layer* is also a must-have feature since it defines the communication among system end-users by distributing the transactions and block data among the accessible peers in the network.
- The *application layer* is a must-have feature since it introduces capabilities to connect with the current enterprise system.
- The selected blockchain platform must be *resistant to Sybil attack* to prevent peer-to-peer network attacks, in which a node in the network runs multiple identities at the same time – guaranteeing the base level of security and resilience.
- *Smart-contracts* is one of the essential features in a blockchain platform whose objective is to validate the state object changes. It is considered a must-have feature.
- To reduce potential risks, the selected blockchain platform should-have *high maturity* and *high popularity*.
- *High transaction speed* is not a crucial issue in the domain context in question, and therefore can be considered as a should-have feature.

- This project context does not require cryptocurrencies, and consequently, consensus mechanisms designed for cryptocurrencies like *proof-of-work* or *proof-of-stake* were considered will not have features. The same applies to cryptocurrency tokens. On the other hand, since the consensus mechanism was not yet decided, the remaining consensus mechanisms – *practical Byzantine fault tolerance, federated Byzantine fault tolerance and delegated Byzantine fault tolerance* – were prioritised as could-have features.

In Table 2, shows the summary of the blockchain features and their prioritization according to the MoSCoW technique.

Table 2. Blockchain features prioritization

MoSCoW	**Blockchain features**
Must-Have	Application layer Network layer Protocol layer Enterprise system integration Permissioned Private Smart contracts Sybil attack resistant
Should-Have	Popularity in the market Technology maturity Transaction speed
Could-Have	Delegated Byzantine fault tolerance Practical Byzantine fault tolerance Federated Byzantine fault tolerance
Will not-Have	Directed Acyclic Graph

	Permissionless Public Proof-of-stake Proof-of-work

After evaluating the selected blockchain features and their prioritization, the DSS showed that the feasible blockchain solutions are R3 Corda, Hyperledger, HydraChain, Symbiont, Chain and JPMorgan Quorum. The decision structure for this decision model can be found in Attachment C.

Figure 10 shows the final report extracted from the framework also shows the impacts of the feasible platforms on market positioning.

	Innovation	Popularity in the market	Technology Maturity	Transaction Speed
R3 Corda	10	10	10	10
Hyperledger	10	10	10	10
HydraChain	2	2	2	5
Symbiont	2	2	5	10
Chain	5	2	5	10
JPMorgan Quorum	5	5	5	10

Figure 10. Non-Boolean blockchain features

All the features presented in Figure 10 are non-Boolean and therefore have a different value when compared to the Boolean functions. The assigned values to this non-Boolean blockchain features, for a specific blockchain platform, were assigned by blockchain experts based on several predefined parameters (Farshidi et al., 2020).

From the market positioning of the feasible blockchain platforms, it is possible to see that only R3 Corda and Hyperledger have higher values for the non-Boolean features and therefore, are the only ones to be considered.

3.1 R3 Corda

Corda is a permissioned and private distributed ledger technology created by R3 - a technology company founded in 2014.

Corda platform offers (Brown, 2019):

- Privacy. Data transaction between parties is private, and only the participants have access to the details of the transaction,
- Interoperability. All versions of Corda – free and commercial – can interoperate with each other.
- Flexibility. Easy to migrate from Corda to Corda Enterprise, as business requirements evolve.
- Immutability. Events recorded on the ledger are final and immutable.

Corda also removes costly friction in business by enabling institutions to transact directly using smart contracts and can easily be integrated with the existing systems, including databases, message queues, and the Java Virtual Machine (Corda, 2019a). It also supports billions of transactions daily across industries (Brown, 2019).

Corda applications can be built inside Intel SGX technology that can help transactions history in the ledger to be encrypted and private (R. Corda, 2019).

Transactions on Corda are not periodically batch up needing for confirmation, into a block, as other blockchain platforms do. Instead, Corda confirms each transaction in real-time, and there is no need to wait for other transactions to come along. This feature of Corda increases privacy and scalability, and it also makes this platform a blockchain and a not blockchain platform (R. Corda, 2019).

This blockchain platform also differs from the others in terms of transactions broadcasting. In Corda, transactions do not get broadcasted to every peer in the network. Instead, the data of the transaction is only accessible to the participants of that transaction. The complete database

– the union of all transactions of all participants over all the time – only exists in a fragmented form, split across the various peers of the network (Corda, 2019b).

This broadcasting particularity brings a considerable boost to the distributed ledger performance since the data does not need to be shared with all members of the network and, consequently, each node only needs to process a tiny fraction of the data (Corda, 2019b).

To ensure consistency in a system, where not all data is visible to all participants, Corda relies on secure cryptographic hashes to identify parties and data, and to link states to previous versions, to provide chains of provenance (Brown, 2019).

In Corda, nodes are backed by a relational database and data placed in the ledger can be queried using SQL (Hearn & Brown, 2019).

Each node has an identity certificate which contains the name of the node that represents the public key or IP address. Communications between nodes are restricted, providing a form of abuse control in which abusive parties can be expelled from the network. An attacker can even issue a certificate for a pre-existing name, but with a new key own by himself, and would not still be allowed to edit or steal any of the existing data. This security of the pre-existing data is possible because, in Corda's design, each state contains a key that cannot be changed after it is created (Hearn & Brown, 2019).

Business logic in Corda is enforced through smart contract code. A smart contract is a pure function whose responsibility is to accept or reject a proposed transaction. Each state object specifies the task that must be executed by any transaction to consume or create a type of state (Brown, 2019).

Corda network is built upon technologies such as Java Virtual Machine and SQL, which supports the use of Cloud systems. This way, it is possible to mitigate the effects of multiple components failures.

Corda uses an AMQP (Advanced Message Queuing Protocol) over TLS (Transport Layer Security) that grantees the routing and the enqueue of messages, even in the case of nodes unavailability. In the case of inaccessibility of a node, the messages queue up and retry until delivery (Corda).

3.1.1 Consensus

In Corda, notary clusters provide ordering and timestamping of transactions. A notary is expected to be composed of multiple parties who use a crash or byzantine fault-tolerant consensus algorithm (Hearn, 2019).

In Corda, updates to the ledger are accomplished using transactions, that consume existing state objects and produce new ones – enabling the creation of the chain of provenance. Notaries accept transactions submitted to them by returning a signature over the transaction or reject the transaction stating that a double-spend has occurred.

Whilst it would be convenient to use a BFT algorithm for notary service, there is no requirement to do so and, this way, Corda is not tied to any consensus algorithm (Hearn, 2019).

The notary service verifies two things in the transaction:

- Transaction validation. A Corda transaction is considered valid if the contract code is verified successfully and if the transaction has the required signatures. Additionally, it is necessary to confirm that any transaction to which the new transactions refers to are also valid.

  Transaction validation is performed only by the nodes associated with the transaction, which means that the transaction data is only shared by the nodes that require to see the transaction.

- Transaction uniqueness. Corda checks the uniqueness of a transaction by ensuring that no previous transaction has been consumed with the same state as the transaction in question. This uniqueness services do not require to see the full contents of any transactions, which improves the privacy of the system.

The transaction validation only occurs when a notary provides validity consensus. Otherwise, it is considered a non-validating notary and will only check for transaction uniqueness.

3.1.2 Smart-contracts and programming language

In Corda, the contracts are Turing-complete and can be written in any ordinary programming language that targets the Java Virtual Machine (Hearn & Brown, 2019). The platform is written entirely in Kotlin, a programming language created by JetBrains and design for JVM, JavaScript,

Android and Native. Kotlin can be used in object-oriented programming and functional programming or with a mix of both (Lang, 2016).

3.2 Hyperledger Fabric

Hyperledger Fabric is a permissioned platform that supports distributed ledger solutions, for industrial use case requirements, providing a secure and robust architecture used to achieve confidentiality, flexibility, resiliency, and scalability (Hyperledger, 2018). Hosted by the Linux Foundation in 2015 is an open-source system for deploying and operating permissioned blockchains (Androulaki et al., 2018).

Hyperledger Fabric is cryptocurrency-agnostic, which means that it will never issue its cryptocurrency. Hyperledger exists to create blockchain software for enterprises and will not administer any cryptocurrency (Hyperledger, 2018).

Fabric introduced the *execute-order-validate* blockchain architecture to oppose the limitations of the currently used architecture *order-execute* that first orders the transaction and then, executes the transaction in the same order in all peers. The limitations of this architecture are (Androulaki et al., 2018):

- Sequential execution. In an *order-execute* architecture, all peers execute the transactions sequentially, which limits the effective throughput of the blockchain.
- Non-deterministic code. Operations executed after consensus must be deterministic, providing all peers in the network with the same transaction state. Universal languages have several problems ensuring the deterministic execution of smart contracts. Only one non-deterministic smart contract, created with malicious intent, is enough to interrupt the whole blockchain.
- Confidentiality of execution. Permissioned systems often run all smart contracts on all peers; even though cryptographic techniques can help achieve privacy, this often comes with a considerable overhead making it not feasible in practice.

With the new *execute-order-validate* architecture, a distributed application for Fabric consists of two parts (Androulaki et al., 2018):

- A central part that is called a *chaincode* which is a smart contract that implements the application logic and runs during the *execution* phase.

- The second part consists of an *endorsement policy* that acts as a static library for the transaction validation, is specified in the chaincode and evaluated in the *validation* phase.

In Fabric's architecture, a transaction sent by the client began firstly to be *executed* by the specific peers of the endorsement policy. This phase output is recorded, and the transaction enters in the *ordering* phase, which uses the consensus protocol to produce an ordered sequence of transactions grouped in blocks. Transactions are then broadcasted to all peers, and each *validates* the state changes from the transactions taking into consideration the endorsement policy, guarantying the consistency of the execution. This validation is deterministic, ensuring that all peers that the same state of the ledger (Androulaki et al., 2018).

In Hyperledger Fabric is possible to create channels between peers which allows them to keep a separated ledger of transactions (Androulaki et al., 2018).

Each peer of the blockchain maintains a copy of the ledger in a key-value store which is a type of a nonrelational database (Androulaki et al., 2018).

3.2.1 Consensus

Unlike almost blockchain systems that have hard-coded consensus mechanisms, Fabric introduced for the first-time pluggable consensus. They defend that the agreement must be tailored with the necessities of the system. The BFT consensus should be reconfigurable and adapt dynamically to a changing environment. Moreover, there can be cases where it may be better to replace the BFT consensus with an alternative like Paxos (Androulaki et al., 2018).

3.2.2 Smart-contracts and programming language

Hyperledger Fabric allows the execution of distributed applications written in general-purpose programming languages, like Java, Go or Node.js (Fabric, 2020).

Chaincode supports plugins for adding new chaincode programming languages and currently, the languages that are supported are Go, Java and Node (Androulaki et al., 2018).

3.3 Distributed Ledger Applications

This section presents distributed ledger applications that are currently using Corda or Hyperledger Fabric and are already in production. These applications are for the insurance sector, and the goal of this section is to explore which technology is leading the market.

OpenIDL (open Insurance Data Link) is an open blockchain network that streamlines regulatory reporting and provides new insights for insurers while enhancing timeliness, accuracy, and value for regulators (American, 2019).

Governed by the American Association of Insurance Services (AAIS) openIDL is the first open blockchain platform that enables the efficient, secure, and permission-based collection and sharing of statistical data (policy, premium, claims and loss experience data) (American, 2019).

AAIS is a national insurance advisory not-for-profit organization in the United States governed by its Member companies (American). AAIS serves as openIDL administrator, providing unbiased governance for the blockchain platform within existing legal and regulatory frameworks. OpenIDL works with IBM Blockchain to radically transform insurance operations with faster verifiable data exchanges, a foundation of transparency, and transactions underpinned with pervasive security and trust. IBM is recognised as the blockchain provider (Juniper, 2017), and is currently working with hundreds of clients across the supply chain, government, financial services, retail, media, and healthcare to implement blockchain applications, and operates some networks running live and in production (American, 2019).

OpenIDL is built on top of the Linux Foundation Hyperledger Fabric permissioned blockchain (American, 2019).

Another Hyperledger Fabric initiative powered by IBM along with Marsh - a leading insurance broker and risk management company – is the opportunity, using blockchain technology, to automate the proof of insurance process and create an immutable record accessible by all network members (IBM, 2019). Launched initially as a proof of concept in April 2018, Marsh's blockchain was rollout in April 2019. This initial rollout allows US commercial clients to quickly search, view and re-issue their certificates of insurance (Marsh, 2019).

Ernst & Young (EY) and Guardtime, in collaboration with other insurance industry leaders, present **Insurwave**, is the world's first blockchain solution for marine insurance, in production with Maersk as the first client, since 2018. The platform uses Corda and is hosted by Microsoft Azure. Insurwave integrates and secures the streams of disparate data sources involved in insuring shipments around the world (Insurwave, 2019).

This platform provides real-time information on the location and conditions of the ship and safety conditions for both insurers and insureds, leading to accuracy and fair underwriting and pricing. In the eventuality of a ship enter a high-risk area, such as war zones, the program detects and factors it into underwriting and pricing calculations (C. Insights, 2019).

B3i, founded in October 2016, is a Blockchain Insurance Industry Initiative supported by twenty major insurance industry investors and over forty companies (Alessandro, 2018). B3i is building distributed ledger applications to address critical insurance needs removing the friction in risk transfer (B3i).

Moving away from Hyperledger Fabric, B3i selected Corda to build its applications and business network in 2018, after completing a review of the available open-source blockchain technologies that better suited the enterprise strategy (Walid, 2019). The study revealed that the Corda platform offers the best-distributed ledger solution, "providing a solid foundation for B3i to efficiently deliver business value to its clients" (Charley, 2018).

In 2019, B3i announced the release of the latest version of its Corda application – Property Catastrophe Excess of Loss Reinsurance – designed for customers to negotiate and place agreements (Walid, 2019).

MetLife, a leading global insurance provider company, also join R3 Corda consortium in 2016 (Pete, 2016). This global insurer launched the world's first parametric insurance solution for gestational diabetes in Singapore (Zia, 2019).

4 Solution

This chapter focuses on the software design of the proof-of-concept application describing the software in three parts: the software requirements specification, the domain design, and the architectural design.

4.1 Software Requirements Specification

Software requirements are the description of the behaviour of the software to be developed. It is a software engineering process that studies the creation, analysis, development, and maintenance of the requirements (both functional and non-functional) that must be satisfied by the system to solve a problem.

This section includes the identification of the involved actors, the use cases description - identifying the interactions between the actors and the system – and lastly, the non-functional requirements that represent the constraint on the design or implementation, like operational costs, reliability and robustness, for example.

4.1.1 Actors

For this proof-of-concept application, there are two types of actors – the ones who interact directly with the process and the ones who interact indirectly with the process.

The insurees (the company and their workers) do not directly intervene in the process but are essential for it to work.

On the other hand, the hospital and the insurance company are the direct actors in the process. The hospital is responsible for submitting the request of insurance and the insurance company for accepting/rejecting their coverage.

4.1.2 Functional Requirements

Functional requirements are described as the functionalities that the software must offer. Therefore, functional requirements are the features that allow the system to function as it is intended.

For the proof-of-concept, the functional requirements gathered in the software requirements specification can be seen in Figure 11 as a use case diagram.

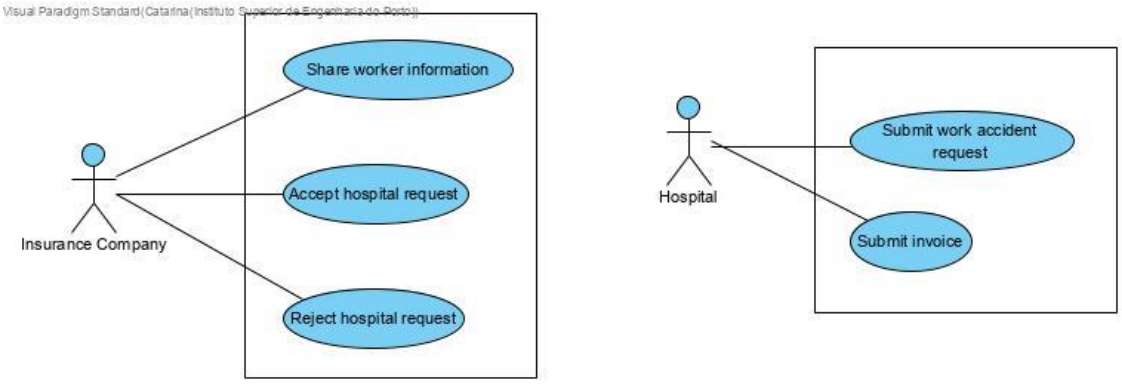


Figure 11. Use case diagram

Listed below, are the functional requirements in detail.

Table 3. Requirement 1 - Share worker information

Requirement No.	**1**
Name	Share worker information
Description	Insurance company shares the worker insurance information with the hospitals that are included in the insurance contract.

Actors	Insurance Company
Preconditions	A new worker is registered in the insurance company
Postconditions	The hospitals can now submit a request when needed

Table 4. Requirement 2 - Submit worker accident request

Requirement No.	**2**
Name	Submit worker accident request
Description	When an injured worker goes to the hospital, the hospital fills the request for the insurance company to evaluate if that treatment can be covered by the insurance policy or not.
Actors	Hospital
Preconditions	A worker is already registered in the hospital
Postconditions	A message is sent to the insurance company

Table 5. Requirement 3 - Accept hospital request

Requirement No.	**3**
Name	Accept hospital request
Description	The insurance company accepts the worker accident request made by the hospital
Actors	Insurance Company
Preconditions	The input state must be in a proposed state
Postconditions	The output state is in an acceptable state, and a message is sent to the hospital

Table 6. Requirement 4 - Reject hospital request

Requirement No.	**4**
Name	Reject hospital request
Description	Insurance company rejects the worker accident request made by the hospital
Actors	Insurance Company
Preconditions	The input state must be in proposal state
Postconditions	The output state is in a rejected state, and a message is sent to the hospital

Table 7. Requirement 5 - Submit invoice

Requirement No.	**5**
Name	Submit invoice
Description	If the insurance company accepts the request, the hospital is notified and can create an invoice to send to the insurance company
Actors	Hospital
Preconditions	The input state must be in accepted state
Postconditions	None

4.1.3 Non-Functional Requirements

Non-functional requirements are a critical part of the software development as they describe the quality attributes of the system. In this dissertation's context, there are some concerns regarding data privacy and security, that can be seen in Table 8. This PoC application does not consider General Data Protection Regulation concerns since it would be necessary to work with a professional that understands the law. GDPR is deemed to be future work.

Table 8. Non-Functional Requirements

Requirement	Description
Privacy	Only the parties involved in the transaction have the permission of seeing the data. The privacy of the data must always be safeguarded.
Data security	The owners must approve all changes made to any data. An update to the claim status must be approved by either the hospital and the insurance company.
Auditability	The application must have the ability to show what happened, who did it and when it was done.

4.2 The Platform for Insurance Information Sharing

As described in the sections 3.1 and 3.2, Corda and Hyperledger are both permissioned distributed ledgers that offer privacy, performance and fine-grained access control. While Fabric aims to interconnect different business sectors, Corda mainly focuses on helping financial entities to work together.

However, in this dissertation context, there are four differences between both blockchain platforms that are crucial to the success of the application.

The first difference is the type of database used for each platform. Corda uses a relational database that can use SQL queries, while Hyperledger uses a non-relational database with key/value store, which means that the data stored in the database is associated with a unique key that is used to query or filter the database. In the context of this dissertation, the worker's compensation process workflow must be submitted several times until it is considered closed. This way, it is necessary that all presented processes, associated with the same incident, can be related to each other. Which means that having a relational database would be beneficial since it is possible to have relationships and a predefined schema.

The second distinction is transactions broadcasting. In Hyperledger, by default, the transactions are broadcasted globally, and gossip networks used to propagate data. It is also possible to create individual communication channels between two or more members for transacting

private and confidential transactions. This concept, for this dissertation's context, would not be beneficial since we would have to create several different channels which could potentially become unmanageable and unscalable.

Corda, on the other hand, uses point-to-point messages and sends them only on a need to know basis (also call lazy propagation), which is precisely the kind of distribution this project looks for since would help to keep the transactions private and confidential.

Another disparity between the technologies is related to transaction propagation and is concerning data storage. Since each Corda node only stores the transaction it is involved in, Corda will typically store less data than a Fabric node in which the channel has more than two members.

The last drawback for Hyperledger Fabric is the way contracts are managed. If it is necessary to add a participant into a contract and prevent them to see the history of it, it must be created a new contract to support that. It not possible to add a participant to a contract without preventing them, so see the full history. On Corda, the existence of unspent transaction outputs allows the ledger to maintain and control who can see the information (Richard, 2017).

For these reasons, in this dissertation project, Corda blockchain platform was chosen to be used in the implementation phase.

4.3 Domain Modeling

The domain model is an artefact that represents entities of the domain, their attributes, and their relationship with each other. It is used to communicate with non-technical stakeholders as it employs the domain vocabulary. In Unified Modelling Language (UML), a class diagram is used to represent the domain model.

In Figure 12, the domain model of the proof-of-work application is presented. It is possible to see the four entities, and the respective properties, that are part of the domain:

- *InsuranceState* is the central entity of this domain and represents the fact known by one or more Corda nodes. This entity consists of:
    - Insured value. Represents the policy value, which means, the amount of money that the insurance contract covers.

- Duration. The number of months that the insurance contract is valid.
- Worker. A reference to the worker table, to whom the insurance contract belongs.
- Claims. A list of references to the work accidents claims of that worker.

- A *Claim* represents the request to the insurance company for coverage or compensation for the covered loss. The claim has:
  - Claim number. The number that identifies a claim.
  - Claim description.
  - Claim amount. The amount of money that was spent on that claim.
  - Internal policy Number. The policy number of the hospital.
  - Accident date. The date when the accident occurs.
  - Episode date. The data when the treatment occurs.
  - Accident type. Property that defined the type of the accident, that in this case, can only be work accident.
  - Module. Where the treatment took place. It can be the urgency, day hospital, external consult, internment, or operating room.
  - Claim status. Represents the state of the claim, that can be a proposal, accepted or rejected.
  - Proposer. The node that submit the claim.
  - Proposee. The node that will receive the claim.
- *InsuranceDetail* represents an entity that is responsible for collect data about the insurance company. This entity is created only in the event of a claim being accepted, where the hospital needs to know more information about the insurance company to be able to create the invoice. This entity is represented with:
  - Insurance company number. The number that identifies the insurance company.
  - Insurance company policy number. The insurance policy number of the insurance company.

- Field. The field of the insurance contract that in this case, is a work accident.

- *Worker* entity represents the insured worker with all the necessary information:
  - Policy number. The number of insurance contract policy.
  - Worker name.
  - Health number. The unique health national number of the worker.
  - Policyholder. The company that is being insured.

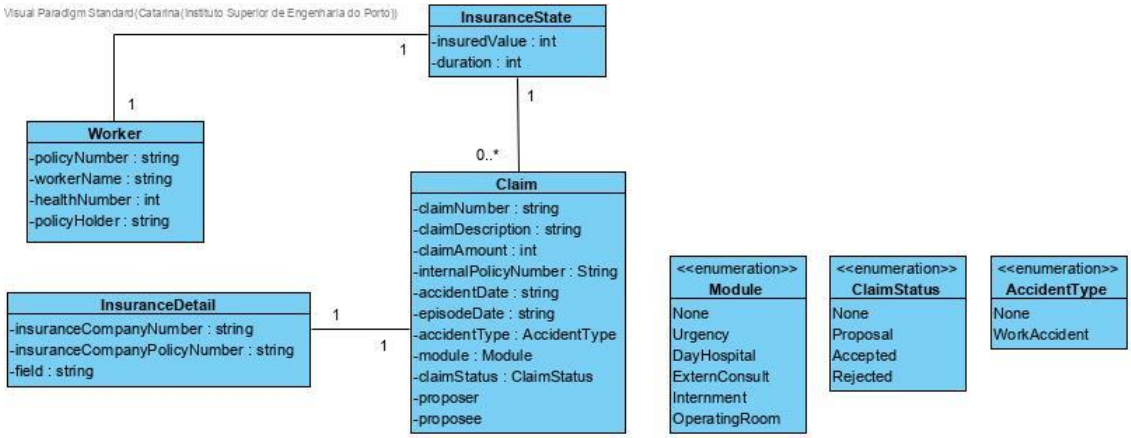


Figure 12. Domain Model Diagram

4.4 Corda Design

To develop a CorDapp some concepts need to be understood first (Corda):

- States - represent shared facts on the ledger.
- Transactions - update the ledger states.
- Contracts - govern how states can evolve.
- Flows - interactions that may occur between parties to achieve consensus.

This section presents these core concepts to be easier to understand the structure of the application.

4.4.1 States

A state is an immutable object that represents a fact known by one or more Corda nodes. Being immutable, for the states to be updated, is necessary to create a new version of the state representing the new state and mark the existing state as historic (Corda).

All these transactions are stores in the vault. Each node on the network maintains a vault which is a database to track all the current and historical states.

Along with information about the fact itself, the state contains a reference to the contract to which it belongs. This association can be seen in Code Snippet 2 with the tag @BelongsToContract, along with the information about the worker compensation insurance state.

4.4.2 Transactions

Transactions evolve the ledge over time by updating transactions, which means to mark zero or more existing states as historical (the transaction input) and producing zero or more states (the transaction output). When creating a new transaction, the input state already exists since it was the valid output state from the previously created transaction. The output state does not exist and needs to be created by the proposer or proposers of the transaction. Both these states (input and output) are included in the proposed transaction by reference (Corda).

To be a valid transaction and be able to update the ledger, the transaction proposal must receive the signatures of all the required signers, becoming committed. Besides the signatures, a transaction also must be contractually valid to be committed.

4.4.3 Contracts

As previously seen, each transaction state specifies a contract type. This contract verifies the input and output state of a transaction to make sure it is contractually valid. Contract execution is deterministic and can be written in Java or Kotlin (Corda).

Contracts in Corda are representations of real-life contracts where condition need to be satisfied to meet the contractual validities. In order to be valid, a transaction not only needs to be contractually valid, but it also needs to be signed by all required signers (Corda).

All Contract classes in Corda implement the Contract interface that has a "*verify*" method that receives the ledger transaction that represents a state transition and ensures the inputs/outputs/commands of that transaction are valid. This method throws an exception if there is any problem that should prevent the state transition. In case of a valid transaction, this method does not return anything.

4.4.4 Flows

A flow is a sequence of steps that specify how a node can achieve the ledger update. The communication between nodes is made exclusively with these flows and is done on a point-to-point basis, contrarily from other Distributed Ledger Technologies, that broadcast the transactions to the entire network. This point-to-point communication requires participants to specify precisely the information that needs to be sent, along with which counterparties that are involved in the transaction (Corda).

A flow is a sequence of steps that tells a node how to achieve a specific ledger update. Each flow describes a sequence of actions for the node to perform. In Figure 13, it is possible to see an example Corda flow where one node – Node A – is attempting to perform a transaction to update the ledger. This flow starts by Node A creating, verifying, and signing a transaction and send it to another node – Node B. Node B then verifies interests and sign the transaction before sending it back to Node A. It then tells Node A to send the transaction on to the notary pool to provide unique consensus. Notary pool signs the transaction before sending it back to Node A. The notary pool tells Node A to record the transaction and send it to Node B, who will also record the transaction.

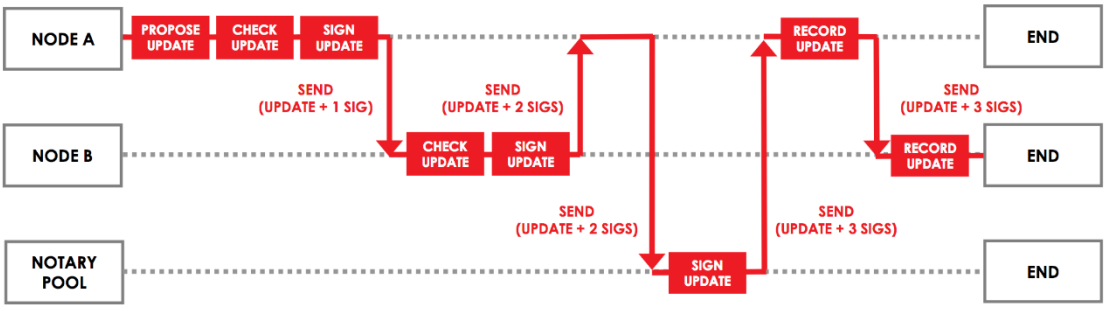


Figure 13. Corda base flow

Source: (Corda, 2018a)

4.5 Architectural Design

The architectural design of an application is a complex activity that consists in the transformation of the requirements specification into an application architecture.

4.5.1 Solution Architectural

In Corda, each of the ledger's node is internally architectured, as shown in Figure 14. This figure shows the core elements of the architecture of a Corda node. At the top is the worker compensation insurance CorDapp. This element connects to the service hub, that is the starting point for most operations that can be performed inside the node. The service hub can be accessed through the flow extension class 'FlowLogic'. The node also has an RPC interface to interact with the node's owner – CordaRPCOps. The node also has the vault and storage service connected to an H2 database. Another visible element is the network interface to communicate with other nodes, using messaging—finally, the client component to host the webserver and the RPC client.

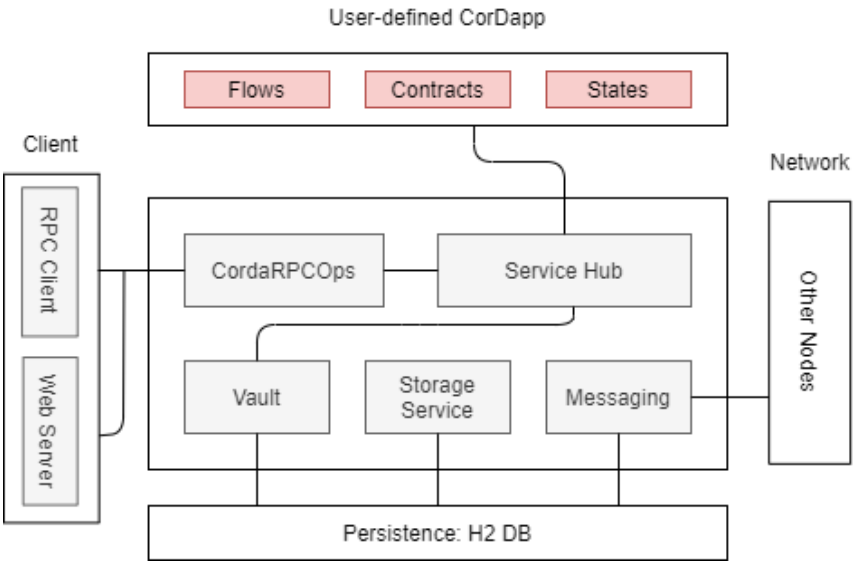


Figure 14. Corda Node Architecture

The network interface is used to handle the communications between the node itself and other nodes on the network. This communication between nodes occurs as part of running a flow which means that the node's owner does not interact with other nodes network directly (Corda).

The interactions with other nodes are made via remote procedure calls (RPC) which hides the details of the network communication.

The service hub is necessary for the node can have access to a set of services that are used, during the execution of the communication flow, to coordinate ledger updates. The primary services provided by the hub are (Corda):

- Information about other nodes on the network and the services they provide.
- Access to the persistence layer.
- Access to, and generation of the node public-private keypairs.
- Information about the node itself.

Corda was designed to be an application layer on top of existing business infrastructure. It is doubtful, within a corporation, that the user interacts directly with the node. Instead, the user will reach the node indirectly, communicating through RPC client making Corda web server agnostic, encouraging firms to use the technologies that might better suit the business needs.

For this proof-of-concept application, each node has a stand-alone, client-side in **Java Spring Boot Web Server**. Spring Boot is an open-source java-based framework used to build stand-alone and production-ready spring applications. There are many benefits of using Java Spring Boot, such as:

- Avoid complex XML configurations.
- Reduce development time.
- Eases dependency management.
- Allows the use of Annotations.

Figure 15 shows the architectural view of a network of Corda nodes couple with their Java Spring Boot web server. In general, a Corda network is composed of:

- Two or more Corda nodes. In the case of this proof-of-concept, there are three nodes, two are representing two different hospitals, and one is the insurance company.
- A network map service responsible for mapping the nodes IP addresses with their identities.
- One or more notary services. Notary service is a network service that provides uniqueness consensus. In the PoC there is only one notary, which is non-validating. This means that the notary will not check the validity of the transaction and will only check for double spend. This way, the notary will not see the full contents of the transaction and compromise privacy.
- Zero or more oracles. Oracles are network services that can obtain information from external sources to the ledger in a trusted way.

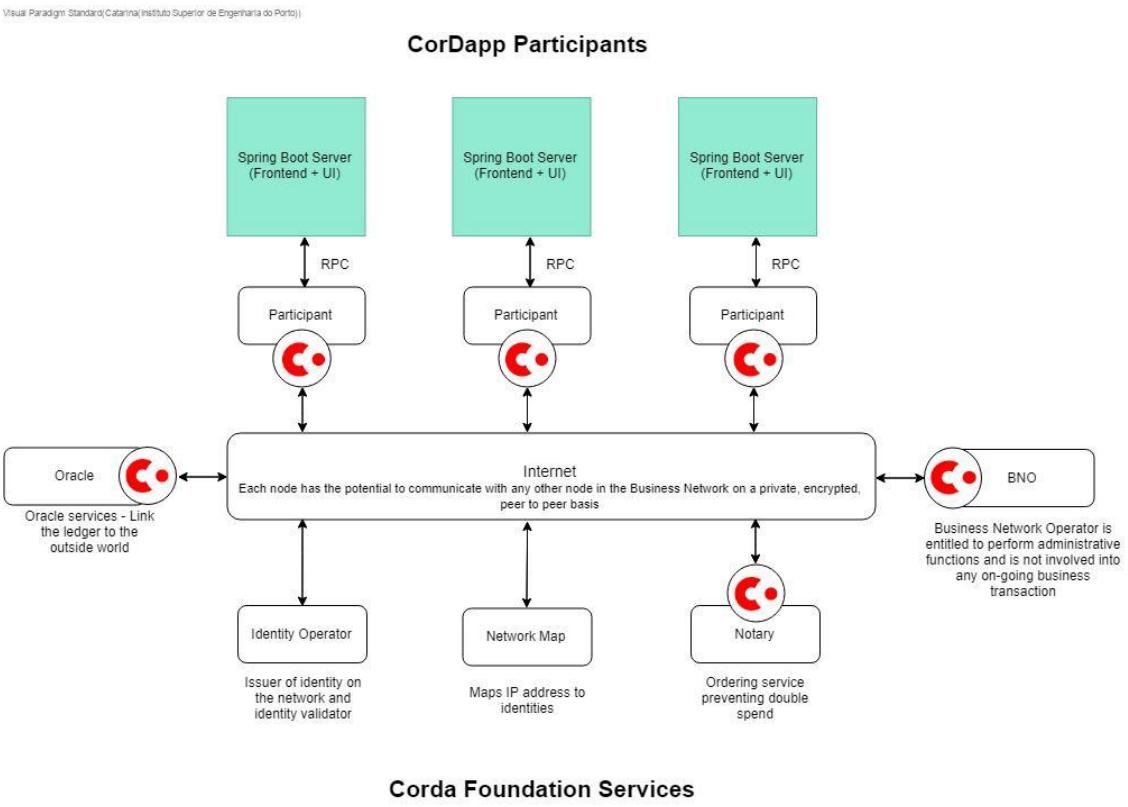


Figure 15. Architectural view of Corda nodes

4.5.2 Deployment Process

As previously stated, a Corda network is composed of one or more notary nodes and two or more Corda nodes. For this proof-of-concept CorDapp, there are just four nodes where one is the notary node, and the other three are the participants of the application.

The notary node:

- Provides a Notary service to validate the double spend.
- Runs the workers compensation insurance Corda application.

The CorDapp participants in this proof-of-concept are the insurance company and two hospitals that work with that insurance company. CorDapp participants nodes:

- Do not provide any service.
- Runs the workers compensation insurance Corda application.
- Have a configured RPC user, which enables the communication with the node.

For a production environment, each node in the Corda network will run in a separated server and will communicate with the other using the network map, which provides a set of reachable nodes.

For this proof-of-concept application, the entire Corda network was deployed on Docker containers. Using a Docker Compose file, it was possible to deploy each node in a different container, using the Corda Docker Official Image. Code Snippet 1 shows the insurance node configuration in Docker Compose file.

```
insureragrs:
    image: corda/corda-zulu-java1.8-4.4
    container_name: insureragrs
    ports:
      - "10006:10201"
      - "2003:2222"
    volumes:
      - ./insureragrs/node.conf:/etc/corda/node.conf
      - ./insureragrs/certificates:/opt/corda/certificates
      - ./insureragrs/persistence:/opt/corda/persistence
      - ./insureragrs/logs:/opt/corda/logs
      - ./shared/cordapps:/opt/corda/cordapps
```

Code Snippet 1. Docker Compose Insurance Configuration

The container configuration is composed with (1) an image that is pulled from docker hub, in this case, is the official Corda image. (2) The name of the container. (3) Ports mapping. The docker image exposes RPC port at 10201 and, in this case, it is being forwarded to 10006 on the local machine. The SSH port, in Corda image in 2222 and is being forwarded to 2001. (4) The volumes that are mounted in the container: the node configuration, the certificates, the persistence (h2 database file), the logs, the worker's compensation insurance CorDapp and the network parameters.

The Docker dashboard, with all containers running, can be seen in Figure 16:

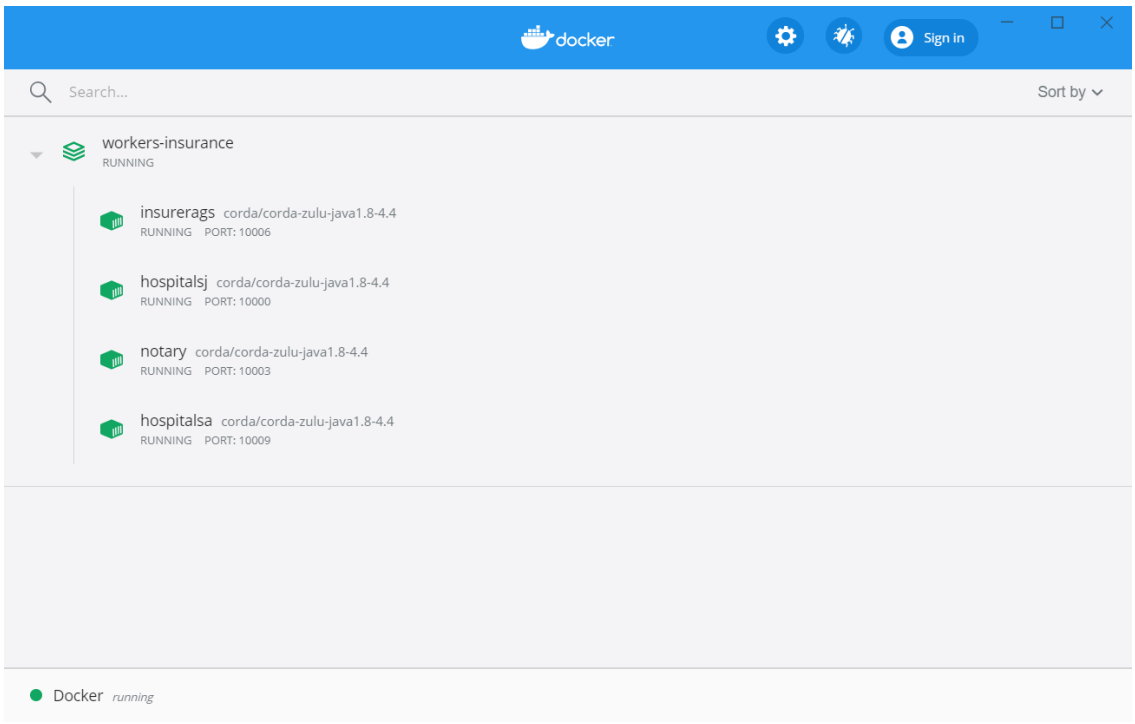


Figure 16. Corda Dashboard

Using the Corda Node Explorer is possible to interact with nodes (Figure 17), and see the node's vault (Figure 18). It is also possible to see the worker insurance network (Figure 19).

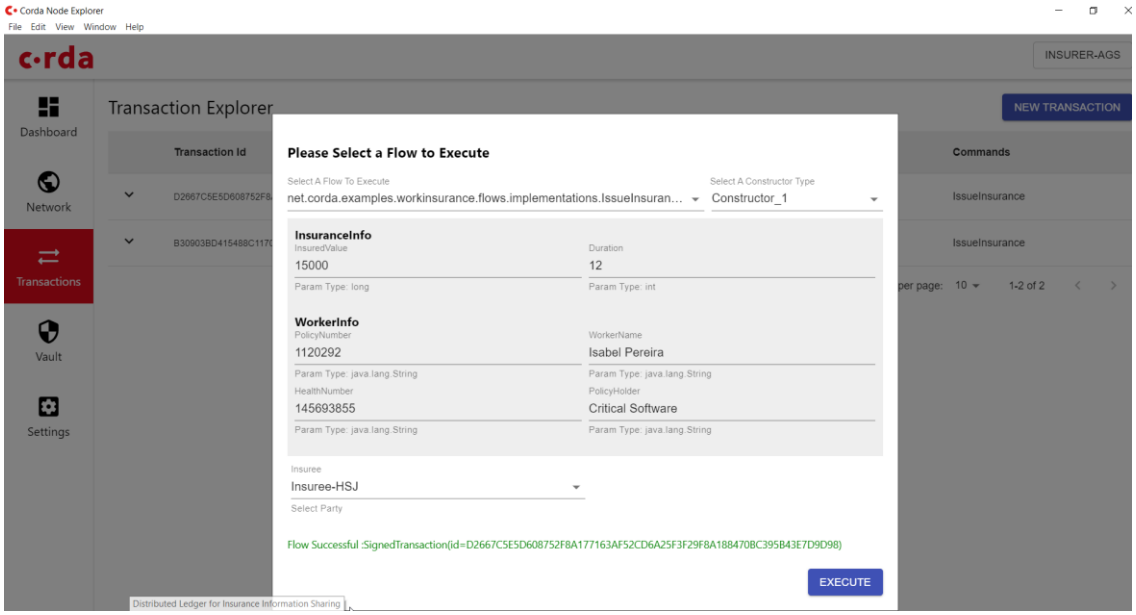


Figure 17. Issue Insurance Corda Node Explorer

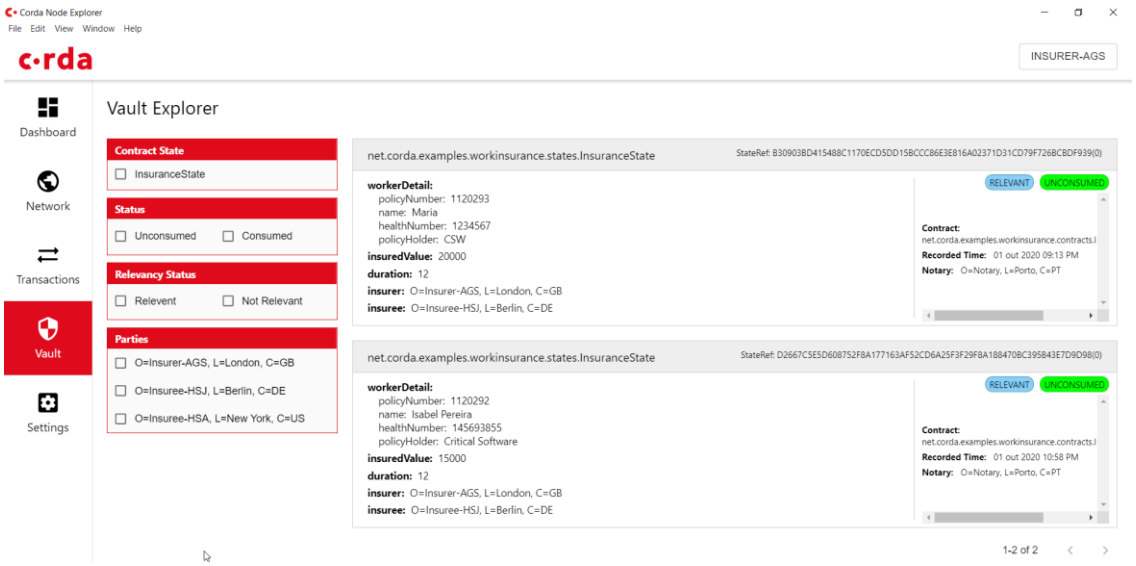


Figure 18. Insurance Vault Explorer

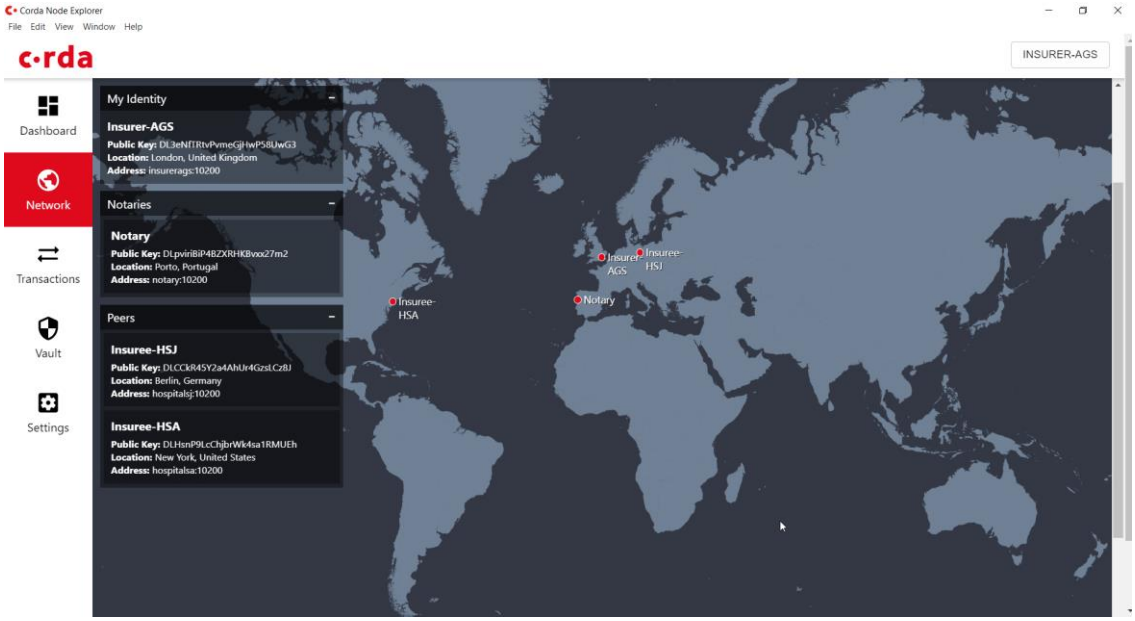


Figure 19. Workers insurance network map

The deployment diagram for the prototype is presented in Figure 20.

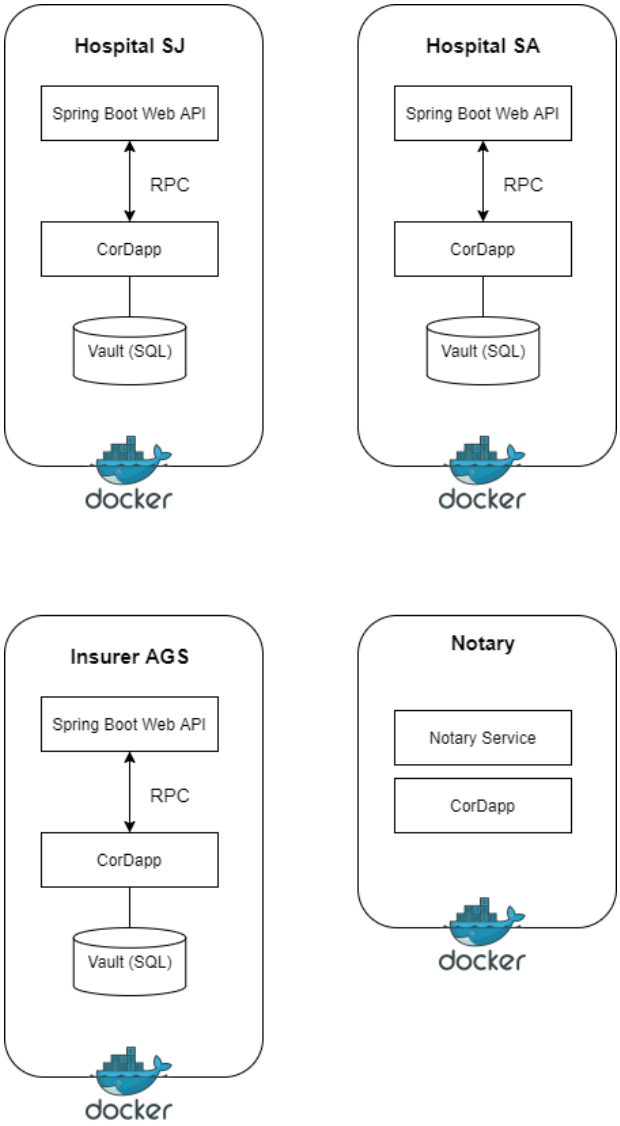


Figure 20. Deployment Diagram

4.6 Implementation

This section presents the details of the worker compensation insurance application. The code for the workers-insurance CorDapp can be found in GitHub:

https://github.com/CatarinaFranco94/workers-insurance.

This PoC application was divided into two parts. The first one was the implementation of the Corda application – server-side, and the second part was to integrate it with a Java Spring Boot Web Server. This section details both parts and additionally, the database access. Corda gives

the possibility of using either Java or Kotlin for the programming language. Since the author was not familiar with Kotlin, this proof-of-concept application was implemented in Java language.

4.6.1 CorDapp

This CorDapp represents a network between an insurance company and two hospitals. To model this network, it was necessary to create a state to represent the insurance, one contract to control the insurance rules, and four flows to interact with the model.

To accurately represent a CorDapp design, Corda created Corda Design Language (CDL) to capture and hence reason about the behaviour of an entire CorDapp. It represents the application design in an accurate, clear and concise way, allowing a common understanding over the system. Figure 21, shows the Corda Design Language view for the PoC, that presents the modelling of the states, their evolution, the required signatures, and the flow of actions.

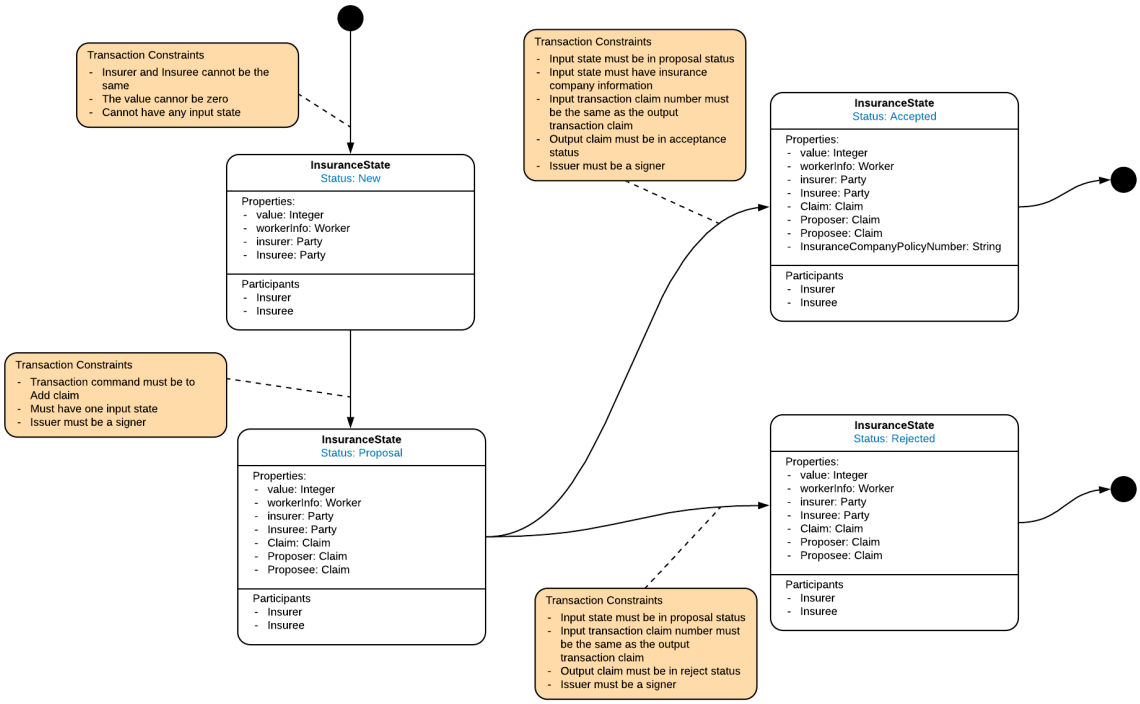


Figure 21. CorDapp Design Language View

As seen in section 4.3, the insurance state is composed of the worker information, the insured value, the duration and, a list of claims. The properties of the insurance state class can be seen

in Code Snippet 2. The *QueryableState* implementation allows the persistence of the state data in a custom database (section 4.6.3).

```
@BelongsToContract(InsuranceContract.class)
public class InsuranceState implements QueryableState {

        // worker information
        private final WorkerDetail workerDetail;

        // insurance value
        private final long insuredValue;

        // insurance duration
        private final int duration;

        // insurance claims made against the insurance policy
        private final List<Claim> claims;
                                ...
```

Code Snippet 2. Insurance State with Associated Contract

As stated previously, each transaction state has a contract type that verifies the input and output state of a transaction to make sure it is contractually valid. In this PoC, there is a contract to verify if the insurance state is valid. Depending on the transaction purpose, there may be different rules to impose a valid transaction. In Corda, *commands* are included in the transaction to indicate the intent of it and consequently affect its validity.

Code Snippet 3 shows the decision statement, in the insurance contract class, that differentiate the validity method for each command - Issue insurance, add a claim, accept a claim and reject a claim. This way is possible to impose different constraints based on the command type.

```
@Override
public void verify(LedgerTransaction tx) {
    CommandWithParties<Commands> command = requireSingleCommand(tx.getCommands(),
InsuranceContract.Commands.class);

    if (command.getValue() instanceof InsuranceContract.Commands.IssueInsurance) {
        verifyInsuranceIssue(tx);

    } else if(command.getValue() instanceof InsuranceContract.Commands.AddClaim) {
        verifyClaimCreation(tx);

    }else if(command.getValue() instanceof InsuranceContract.Commands.AcceptClaim) {
        verifyClaimAcceptance(tx);

    }else if(command.getValue() instanceof InsuranceContract.Commands.RejectClaim){
        verifyClaimReject(tx);

    } else {
        throw new IllegalArgumentException("Unrecognized command");
    }
}
```

Code Snippet 3. Insurance transaction commands

In the workers-insurance application, the insurance contract has some basic rules that need to verify to guarantee the validity of a transaction. Part of the contract can be seen in Code Snippet 4. The contract described in Code Snippet 4 belongs to the insurance state and is triggered when the transaction has an AcceptClaim command. This insurance contract checks if the input state and the output state meet the conditions that confirm the validity of the transaction:

- The transaction must have input states.
- The transaction must have precisely one input state.
- Input transaction must be of type InsuranceState.
- Insurance transaction must have output states.
- Insurance transaction must have exactly one output.
- Output transaction must be of type InsuranceState.
- The insurance transaction must have commands.
- The insurance transaction must have a command of type AcceptClaim command.
- The insurance output transaction must have a claim.
- The insurance output transaction must have the last claim in an Accepted state.
- The insurance output transaction must have insurance detail information (this information is only added in the event of an accepted claim).

- The insurance input transaction must have claims.
- The insurance input transaction must have the last claim in a Proposal state.
- The insurance input transaction cannot have insurance detail information.
- The input transaction must have one claim that matches the claim number of the claim that is being verified.
- The issuer of the transaction must be a required signer. (In this case, the issuer of the transaction is the insurance company – the entity that can approve a claim).

Many more conditions can be added to the contract, but for simplicity proposes, this proof-of-concept only has some basic ones.

```
private void verifyClaimAcceptance(LedgerTransaction tx){
    requireThat(req -> {
        req.using("Insurance transaction must have input states",
(!tx.getInputStates().isEmpty()));
        req.using("Insurance transaction must have one input state",
tx.getInputStates().size() == 1);
        InsuranceState inputState = (InsuranceState)tx.getInput(0);
        req.using("Insurance input transaction must be of type InsuranceState",
inputState instanceof InsuranceState);

        req.using("Insurance transaction must have output states",
(!tx.getOutputStates().isEmpty()));
        req.using("Insurance transaction must have one output state",
tx.getOutputStates().size() == 1);
        InsuranceState outputState = (InsuranceState) tx.getOutput(0);
        req.using("Insurance output transaction must be of type InsuranceState",
outputState instanceof InsuranceState);

        req.using("Insurance transaction must have a command", tx.getCommands().size()
== 1);
        req.using("Insurance transaction command must be a AcceptClaim command",
tx.getCommands().get(0).getValue() instanceof InsuranceContract.Commands.AcceptClaim);

        req.using("Insurance output transaction must have claims",
(!outputState.getClaims().isEmpty()));
        Claim outputClaim = outputState.getClaims().get(outputState.getClaims().size()
- 1 );
        req.using("Output Claim must be in acceptance status",
outputClaim.getClaimStatus().equals(ClaimStatus.Accepted));
        req.using("Output claim must have insurance detail information",
outputClaim.getInsuranceDetail() != null);

        req.using("Insurance input transaction must have claims",
(!inputState.getClaims().isEmpty()));
        List<Claim> inputClaimListWithOutputClaimNumber =
inputState.getClaims().stream()
                .filter(claimList ->
claimList.getClaimNumber().equals(outputClaim.getClaimNumber()))
                .collect(Collectors.toList());
        Claim inputClaim =
inputClaimListWithOutputClaimNumber.get(inputClaimListWithOutputClaimNumber.size() -
1);
        req.using("Input state claim must be in proposal status",
(inputClaim.getClaimStatus().equals(ClaimStatus.Proposal)));
        req.using("Input claim must not have insurance detail information",
inputClaim.getInsuranceDetail() == null);
        req.using("Input State must have at least one claim with the same number as
the output claim", (!inputClaimListWithOutputClaimNumber.isEmpty()));

        req.using("Issuer must be a required signer",
tx.getCommands().get(0).getSigners().contains(outputState.getInsurer().getOwningKey())
);
        return null;
    });
}
```

Code Snippet 4. Insurance acceptance contract

Finally, this workers-insurance implements four different flows:

1. Issue an insurance contract.
2. Issue an insurance claim proposal.
3. Accept the claim proposal.

4. Reject the claim proposal.

Depending on whether the transaction has input or not, the flow has an additional step to fetch the input state from the node vault. The rest of the steps are equals to all flows and intent to:

- Create an output state.
- Build the transaction with the input state, output state and command.
- Verify the transaction (which will run the contract).
- Sign the transaction with the issuer private key, making this transaction immutable.
- Sign the transaction that was previously signed by the issuer, with the counterparty key.
- Call the finality flow, which will get the transaction notarised and recorded automatically by the platform.

Code Snippet 5 shows part of the issue insurance flow class. Like any flow classes, the Issue*InsuranceInitiator* flow has an annotation called *@InitiatingFlow* and extends *FlowLogic*. This combination is required in order to communicate with a counterparty. *FlowLogic* contains one abstract function – *call* – that needs to be implemented by the flow.
The *@StartableByRPC* annotation in each flow class means that the flow can be called from an RPC connection which, as seen previously, is the interface to interact with the Corda node. This annotation is responsible for triggering the flow that will execute the *call* method and subsequently, all the logic inside the function runs.

```
@Suspendable
@Override
public SignedTransaction call() throws FlowException {
    final Party notary =
getServiceHub().getNetworkMapCache().getNotaryIdentities().get(0);

    Party insurer = getOurIdentity();

    WorkerInfo workerInfo = insuranceInfo.getWorkerInfo();
    WorkerDetail workerDetail = new WorkerDetail(workerInfo.getPolicyNumber(),
workerInfo.getName(), workerInfo.getHealthNumber(), workerInfo.getPolicyHolder());

    // Build the insurance output state.
    InsuranceState insurance = new InsuranceState(insuranceInfo.getInsuredValue(),
            insuranceInfo.getDuration(), insurer, insuree, workerDetail,null);

    // Build the transaction
    TransactionBuilder builder = new TransactionBuilder(notary)
        .addOutputState(insurance, InsuranceContract.ID)
        .addCommand(new InsuranceContract.Commands.IssueInsurance(),
ImmutableList.of(insurer.getOwningKey(), insuree.getOwningKey()));

    // Verify the transaction
    builder.verify(getServiceHub());

    FlowSession session = initiateFlow(insuree);

    // We sign the transaction with our private key, making it immutable.
    SignedTransaction signedTransaction =
getServiceHub().signInitialTransaction(builder);

    // The counter party signs the transaction
    SignedTransaction fullySignedTransaction = subFlow(new
CollectSignaturesFlow(signedTransaction, singletonList(session)));

    // We get the transaction notarised and recorded automatically by the platform.
    return subFlow(new FinalityFlow(fullySignedTransaction,
singletonList(session)));
}
```

Code Snippet 5. Issue insurance initiator flow

The response flow class is also implemented extending the FlowLogic but with a newly tagged *@InitiatedBy* which indicates that the response class is designed to be initiated by a counterparty flow. The response class must have a parameter which represents the initiating counterparty. This response flow is also responsible for verifying if the transaction is according to some conditions. If the conditions are met, the node signs the transaction, and the flow ends.

The response class of the issue insurance initiator flow can be seen in Code Snippet 6.

```
@Override
@Suspendable
public Void call() throws FlowException {
    SignedTransaction signedTransaction = subFlow(new
SignTransactionFlow(counterpartySession) {
        @Suspendable
        @Override
        protected void checkTransaction(SignedTransaction stx) throws FlowException {
            requireThat(require -> {
                ContractState output = stx.getTx().getOutputs().get(0).getData();
                require.using("This must be an InsuranceState transaction.", output
instanceof InsuranceState);
                InsuranceState insuranceState = (InsuranceState) output;
                require.using("Transaction must have valid worker detail",
insuranceState.getWorkerDetail() != null);
                return null;
            });
        }
    });
    subFlow(new ReceiveFinalityFlow(counterpartySession, signedTransaction.getId()));
    return null;
}
```

Code Snippet 6. Insurance claim response flow

4.6.2 Spring Boot Web Server

This section provides a simple explanation of how to create a *Java Spring Boot Web Server* that connects to a Corda node via RPC. Spring boot is an open-source Java-based framework that is used to develop a stand-alone spring application. With *Spring Boot* is possible to process and manage REST endpoints, it is possible to use Spring annotations and consequently, increase productivity and reduce the development time. With *Spring Boot,* everything is auto-configured, and no manual configurations are needed.

The entry point of the spring boot application is the class that contains the annotation *@SpringBootApplication* along with the “main” method to run the application.

The interaction between the server and the Corda node needs to establish via an RPC connection. This connection can be accomplished via a class that forms an RPC connection to the Corda node; this class can be found in the “clients” package and is called *NodeRPCConnection*. When invoked, this class returns an RPCConnection containing a proxy that enables RPC operations on the node.

Each API is contained within a *Spring Boot Rest Controller* with the annotation *@RestController*. Within this class are the necessary methods to use the webserver. Code Snippet 7 shows the controller for this proof-of-concept application.

```
@RestController
@RequestMapping("/")
public class Controller {
    private final CordaRPCOps proxy;
    private final static Logger logger = LoggerFactory.getLogger(Controller.class);

    public Controller(NodeRPCConnection rpc) {
        this.proxy = rpc.proxy;
    }

    @PostMapping(value = "/workerInsurance/{insuree}")
    private String workerSale(@RequestBody InsuranceInfo insuranceInfo,
@PathVariable String insuree) {... }

    @PostMapping(value = "/workerInsurance/claim/{policyNumber}")
    private String claim(@RequestBody ClaimInfo claimInfo, @PathVariable String
policyNumber) { ... }

    @PostMapping(value =
"/workerInsurance/acceptanceClaim/{policyNumber}/{claimNumber}/{insuree}")
    private String claimAcceptance(@RequestBody InsuranceDetailInfo
insuranceDetailInfo, @PathVariable String insuree, @PathVariable String
policyNumber, @PathVariable String claimNumber) { ... }

}
```

Code Snippet 7. Webserver Controller

4.6.3 Database Access

By default, nodes store their data in an H2 database that will not be exposed. To configure the node to expose its internal database, it is only necessary to specify the full network address using the h2Settings. This database can be accessed using a socket which can be browsed using any tool that can use JDBC (Java Database Connectivity), drivers.

H2 database for the workers-insurance application can be seen in the following images. In this database, there are three tables:

- Worker_Detail. Saves the data about the worker.

SELECT * FROM WORKER_DETAIL;

ID	HEALTHNUMBER	POLICYHOLDER	POLICYNUMBER	WORKERNAME
58ca31a0781a43f6bf1c9d4901c7acfd	C232ND832	Critical Software	MH014343	António
0a1358b182f040a782a3bda02d75fbec	C232ND832	Critical Software	MH014343	António
c2336a00201d4a519d37570eb7ae8d60	C232ND832	Critical Software	MH014343	António

(3 rows, 1 ms)

Figure 22. Worker_Detail table

- Insurance_Detail. Stores the insurance data; it has the worker identifier and the transaction identifier.

SELECT * FROM INSURANCE_DETAIL ;

OUTPUT_INDEX	TRANSACTION_ID	DURATION	INSUREDVALUE	WORKER_ID	POLICYNUMBER
0	C6A7311036E8E13EBBC8D1A4ED8AEA21EF0F9BCEC4B7592EA1AA3F38B9314A9F	1	20000	58ca31a0781a43f6bf1c9d4901c7acfd	MH014343
0	122158743001CA78A5766558DC764CC61EDC36BBE0AAFDF80C043F75EE7496AE	1	20000	0a1358b182f040a782a3bda02d75fbec	MH014343
0	77D378A1BFA69473999F38BBA3EF28E8D3B356D4163523ADB80C8E09202D9F2B	1	20000	c2336a00201d4a519d37570eb7ae8d60	MH014343

(3 rows, 0 ms)

Figure 23. Insurance_Detail table

- Claim_Detail. Stores data about the claim. For presentation purposes, Figure 24 does not show all the properties of the claim but only the basic ones. In this table, it is possible to see the transaction ID that can be associated with the insurance detail from Figure 23.

SELECT ID, AccidentType, ClaimNumber, ClaimStatus, Transaction_Id, Proposer, proposee FROM CLAIM_DETAIL;

ID	ACCIDENTTYPE	CLAIMNUMBER	CLAIMSTATUS	TRANSACTION_ID	PROPOSER	PROPOSEE
5162009dc805464ab19629fa87c05c66	WorkAccident	C001	Proposal	122158743001CA78A5766558DC764CC61EDC36BBE0AAFDF80C043F75EE7496AE	O=Insuree-HSJ, L=New York, C=US	O=Insurer-AGS, L=London, C=GB
e616da32ff764847a3de779e400297a0	WorkAccident	C001	Proposal	77D378A1BFA69473999F38BBA3EF28E8D3B356D4163523ADB80C8E09202D9F2B	O=Insuree-HSJ, L=New York, C=US	O=Insurer-AGS, L=London, C=GB
eed775608d044cd99dd1a30b85f3b047	WorkAccident	C001	Accepted	77D378A1BFA69473999F38BBA3EF28E8D3B356D4163523ADB80C8E09202D9F2B	O=Insurer-AGS, L=London, C=GB	O=Insuree-HSJ, L=New York, C=US

(3 rows, 0 ms)

Figure 24. Claim_Detail table

The immutability of Corda transactions can be seen in this proof-of-concept database (Figure 22, Figure 23 and Figure 24), where it is possible to see the duplication of information. This repetition is not, in fact, a duplication, but a new transaction, with new data, that updates the ledger.

Figure 22, shows three workers with the same information; this means that the Corda application received three transactions for that worker. In Figure 23 and Figure 24 is possible to see the transaction identifier, and this way, it is possible to relate the transaction with the worker and claims. The steps that originate these data were:

1. A new worker enters the blockchain (Figure 23, first line, with the worker identifier from Figure 22, first line).
2. A new claim registered for this worker, and it is in proposal state (in Figure 23, the second line transaction identifier is related to the first claim in Figure 24).
3. The insurance claim was accepted. In Figure 23, the last entry has a transaction identifier that relates to the second and third line of Figure 24 showing that the transaction claim was first in a proposal state, but evolve to an acceptable state.

The requests made to the localhost webserver can be found in Attachment D.

5 Tests and Solution Validation

When developing software, there must be some form of evaluating if, for a specific timeframe, this software reaches the proposed objectives and goals.

It is necessary to ensure that all blockchain components work correctly and that there is not any defect present in the software. This way, trust can be promoted. The goal of the tests described in the next section is to guarantee that the developed prototype accomplishes both the functional and non-functional requirements.

This prototype goal is to prove that blockchain can be an excellent alternative to centralized systems and still maintain the security, availability, and reliability. This way is possible to reduce the responsibilities of the central authority distributing it among the interested parties. In this dissertation context, the central authority is APS that will not intervene in the process and will only serve as a provider of a business case that can benefit from a distributed ledger technology.

5.1 Tests

Software testing is one of the most critical phases of a software development system. In a decentralized system, software testing is especially critical because, without the central authority, it is more complicated to roll out fixes or oversee corrections.

Continuous testing is an essential pillar, in every software product, that establishes trust about the quality of the developed software. Continuous testing should follow the best practice-based suite of testing approaches. For a blockchain project, the recommended tests include (QA, 2018):

- Smart Contract Testing. Smart Contract testing is mainly about performing detailed functional testing of business logic and process.
- Infrastructure Testing. This type of verifies, whether the end-to-end Blockchain core network and its various components are operating as expected.
- Functional Testing. Functional Testing validates the Blockchain, its subsystems and the associated business processes:
    - Network setup.
    - Node creation.
    - Message injection.
    - The consensus among nodes.
    - Acceptance of new messages.
    - Validation of the expected behaviour.
- Integration Testing. Integration tests are meant to verify the integration of nodes with the outside system. They should validate:
    - The communication between nodes in the blockchain.
    - The message content.
    - Integration with third-party applications and APIs.
- Security Testing. A Blockchain network is only as secure as its infrastructure. Security validation should be performed at a transaction, block, and network-level for Blockchain networks, including:
    - Identity/Access testing for users and administrators.
    - Data integrity validation using tests that assess the ease of modifying data and assess its impact on application behaviour.
    - Perform encryption security and safety standards tests to prevent any misuse and misappropriation.

- Performance Testing. Performance tests are meant to verify the responsiveness of the system. This type of tests enables the identification of performance bottlenecks and allow the definition of a metric to adjust the system.

5.1.1 Performed Tests

The unit tests were developed for the contract, the state, and the flows.

The contract unit tests were made specially to verify:

- The number of inputs and outputs in the transaction.
- The transaction output type and properties.
- The number of commands in the transaction.
- Type of command in the transaction.
- The presence of the issuer signature.

All the tests have at least two variations, one to test the success and other the failure. In Code Snippet 8, it is possible to see an example of a test to the *AcceptClaimContract* that checks the number of outputs in the transaction. In the first case, the transaction has two outputs and will fail, so the assert will verify that the transaction is adequately failing. In the second case, the transaction has the correct number of outputs, and the assert will verify that the transaction is valid.

```
@Test
public void insuranceContractAcceptClaimCommandRequiresOneOutputInTheTransaction(){
    transaction(ledgerServices, tx -> {
        tx.input(InsuranceContract.ID, insuranceStateClaimProposal);
        // Has two output, will fail.
        tx.output(InsuranceContract.ID, insuranceStateClaimAccepted);
        tx.output(InsuranceContract.ID, insuranceStateClaimAccepted);
        tx.command(Arrays.asList(ags.getPublicKey(), hsj.getPublicKey()), new
InsuranceContract.Commands.AcceptClaim());
        tx.fails();
        return null;
    });

    transaction(ledgerServices, tx -> {
        tx.input(InsuranceContract.ID, insuranceStateClaimProposal);
        // Has one output, will verify
        tx.output(InsuranceContract.ID, insuranceStateClaimAccepted);
        tx.command(Arrays.asList(ags.getPublicKey(), hsj.getPublicKey()), new
InsuranceContract.Commands.AcceptClaim());
        tx.verifies();
```

Code Snippet 8. Accept claim output unit test

The state unit tests were made specially to verify the gets and sets to all the properties. A simple example can be seen in Code Snippet 9.

```
@Test
public void insuranceStateHasTwoParticipantsTheIssuerAndTheInsuree(){
    InsuranceState insuranceState = new InsuranceState(2000, 10, agsInsurance,
hospitalSaoJoao, workerDetail, Collections.emptyList());
    assertEquals(2, insuranceState.getParticipants().size());
    assertTrue(insuranceState.getParticipants().contains(agsInsurance));
    assertTrue(insuranceState.getParticipants().contains(hospitalSaoJoao));
}
```

Code Snippet 9. Insurance state participants test

All the insurance flows, described in section 4.6.1, were unit tested to validate that the transaction constructed by the flow:

- Uses the correct notary.
- Has one insurance state output with the correct claim information.
- Has one output using the correct contract.
- Has one accept claim command.
- Has one command with the issuer and the owner as a signer.
- Has no input attachments or time windows.

Code Snippet 10 shows a test performed to the insurance claim accept flow. This test is validating if both involved parties' signatures are present in the output transaction.

The nodes used in this unit tests were mocked using Corda's test package.

```
@Test
public void
transactionConstructedByFlowHasOneCommandWithTheIssuerAndTheOwnerAsASigners() throws
Exception {
    WorkerInfo workerInfo = new WorkerInfo("policyNr", "Alfredo", "123456", "CSW" );
    InsuranceInfo insuranceInfo = new InsuranceInfo(20000, 20, workerInfo);

    IssueInsuranceFlow.IssueInsuranceInitiator insuranceIssueFlow = new
IssueInsuranceFlow.IssueInsuranceInitiator(insuranceInfo,
b.getInfo().getLegalIdentities().get(0));
    a.startFlow(insuranceIssueFlow);
    network.runNetwork();

    InsuranceClaimFlow.InsuranceClaimInitiator claimIssueFlow = new
InsuranceClaimFlow.InsuranceClaimInitiator(claimInfo, workerInfo.getPolicyNumber());
    b.startFlow(claimIssueFlow);
    network.runNetwork();

    InsuranceDetailInfo insuranceDetailInfo = new InsuranceDetailInfo("CompNr",
"PolNr", "field");
    InsuranceAcceptanceClaimFlow.InsuranceAcceptanceClaimInitiator flow = new
InsuranceAcceptanceClaimFlow.InsuranceAcceptanceClaimInitiator(insuranceDetailInfo,
workerInfo.getPolicyNumber(),claimInfo.getClaimNumber(),b.getInfo().getLegalIdentiti
es().get(0));
    CordaFuture<SignedTransaction> future = a.startFlow(flow);
    network.runNetwork();
    SignedTransaction signedTransaction = future.get();

    assertEquals(1, signedTransaction.getTx().getOutputStates().size());
    Command command = signedTransaction.getTx().getCommands().get(0);

    assertEquals(2, command.getSigners().size());

assertTrue(command.getSigners().contains(a.getInfo().getLegalIdentities().get(0).get
OwningKey()));

assertTrue(command.getSigners().contains(b.getInfo().getLegalIdentities().get(0).get
OwningKey()));
}
```

Code Snippet 10. Insurance claim accept flow test

These unit tests validate the business logic and processes of the network, covering the smart contract testing from the previous section. This tests also validate the expected behaviour of the nodes and their contracts, validating all the conditions of it.

There is also a test for the network setup and node creation that can be seen in Code Snippet 11.

```
public static void main(String[] args) {
    final List<User> rpcUsers =
        ImmutableList.of(new User("user1", "test", ImmutableSet.of("ALL")));

    driver( new DriverParameters()
        .withStartNodesInProcess(true)
        .withWaitForAllNodesToFinish(true),
        dsl -> {
                try {
                    dsl.startNode(
                      new NodeParameters()
                            .withProvidedName(
                              new CordaX500Name("PartyA", "London", "GB"))
                            .withRpcUsers(rpcUsers)).get();
                    dsl.startNode(new NodeParameters()
                            .withProvidedName(
                              new CordaX500Name("PartyB", "New York", "US"))
                            .withRpcUsers(rpcUsers)).get();
                } catch (Throwable e) {
                    System.err.println(
                      "Encountered exception in node startup: " + e.getMessage());
                    e.printStackTrace();
                }

                return null;
        }
);
```

Code Snippet 11. Network setup and node creation test

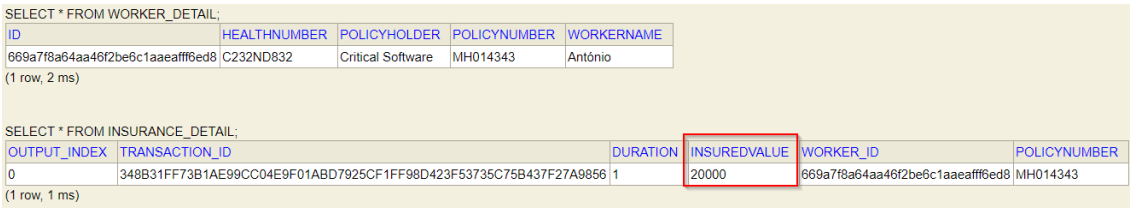
The integration tests were performed to test the communication between simulated nodes using RPC. The asserts check the correctness of the output transaction and, most importantly, the privacy of the transaction. The test uses three running nodes, but only two nodes can be a part of the transaction. The tests check that the third party does not have access to the transaction by checking its vault. An example of an integration test like is described in Code Snippet 12.

```
@Test
public void issueInsuranceFlowTest() {
    driver(new DriverParameters().withIsDebug(true).withStartNodesInProcess(true),
dsl -> {
        // Start a pair of nodes and wait for them both to be ready.
        List<CordaFuture<NodeHandle>> handleFutures = ImmutableList.of(
                dsl.startNode(new
NodeParameters().withProvidedName(insuranceAGS.getName())),
                dsl.startNode(new
NodeParameters().withProvidedName(hospitalA.getName())),
                dsl.startNode(new
NodeParameters().withProvidedName(hospitalB.getName()))
        );

        try {
            NodeHandle partyAHandle = handleFutures.get(0).get();
            NodeHandle partyBHandle = handleFutures.get(1).get();
            NodeHandle partyCHandle = handleFutures.get(2).get();

            Party partyA = partyAHandle.getNodeInfo().getLegalIdentities().get(0);
            Party partyB = partyBHandle.getNodeInfo().getLegalIdentities().get(0);


            WorkerInfo workerInfo = new WorkerInfo("policyNr", "Alfredo", "123456",
"CSW" );
            InsuranceInfo insuranceInfo = new InsuranceInfo(20000, 20, workerInfo);

            // Run issue transaction using rpc
               partyAHandle.getRpc().startTrackedFlowDynamic(IssueInsuranceFlow.Issu
       eInsuranceInitiator.class, insuranceInfo, partyB)
                    .getReturnValue().get();

            // Query Node A
            Vault.Page<InsuranceState> insuranceStateStates_A =
partyAHandle.getRpc().vaultQuery(InsuranceState.class);
            assertEquals(1, insuranceStateStates_A.getStates().size());

            InsuranceState insururanceState_A =
insuranceStateStates_A.getStates().get(0).getState().getData();
            assertEquals(partyA, insururanceState_A.getInsurer());
            assertEquals(partyB, insururanceState_A.getInsuree());
            assertNotNull(insururanceState_A.getWorkerDetail());

            // Query Node B
            Vault.Page<InsuranceState> tokenStates_B =
partyBHandle.getRpc().vaultQuery(InsuranceState.class);
            assertEquals(1, tokenStates_B.getStates().size());

            InsuranceState insuranceState_B =
tokenStates_B.getStates().get(0).getState().getData();
            assertEquals(partyA, insuranceState B.getInsurer());
            assertEquals(partyB, insuranceState_B.getInsuree());
            assertNotNull(insuranceState_B.getWorkerDetail());

            //Query Node C
            Vault.Page<InsuranceState> insuranceStates_C =
partyCHandle.getRpc().vaultQuery(InsuranceState.class);
            assertEquals(0, insuranceStates_C.getStates().size());

        } catch (Exception e) {
            throw new RuntimeException("Caught exception during test: ", e);
        }

        return null;
    });
}
```

Code Snippet 12. Issue insurance flow integration test

Additionally, a manual test to the system was performed locally, to test the blockchain core end-to-end and evaluate its responsiveness. First, a network with four nodes – the notary, an insurance company and two hospitals – was set up. After initiating the nodes and the respective Java Sprint Boot application, a claim was processed between one of the hospitals and the insurance company. The results showed that the intervenient hospital and the insurance company that participate in the transaction had all the history of the claim in their database, while the second hospital database remained empty. In addition, the results concluded that the responsiveness of the system is excellent since transactions can be seen, in the database, within seconds of being sent.

For the security testing, an attempt to tampering the node data was performed, directly in the h2 database server. Initially, a transaction was sent from the insurance company to the hospital indicating the worker information, with the insurance value set to twenty thousand euros (Figure 25). Next, it was performed an update directly to the h2 database server (Figure 26). Finally, a new transaction was created between the hospital and the insurance company, proposing a claim (Figure 27). It is possible to see that the insurance amount, that was initially changed from twenty thousand to three thousand is still present in the first transaction, but the new transaction still has the correct and immutable amount for the insurance value.


SELECT * FROM WORKER_DETAIL;

ID	HEALTHNUMBER	POLICYHOLDER	POLICYNUMBER	WORKERNAME
669a7f8a64aa46f2be6c1aaeafff6ed8	C232ND832	Critical Software	MH014343	António

(1 row, 2 ms)

SELECT * FROM INSURANCE_DETAIL;

OUTPUT_INDEX	TRANSACTION_ID	DURATION	INSUREDVALUE	WORKER_ID	POLICYNUMBER
0	348B31FF73B1AE99CC04E9F01ABD7925CF1FF98D423F53735C75B437F27A9856	1	20000	669a7f8a64aa46f2be6c1aaeafff6ed8	MH014343

(1 row, 1 ms)

Figure 25. Issue Worker Insurance


UPDATE INSURANCE_DETAIL SET INSUREDVALUE = '3000';

SELECT * FROM WORKER_DETAIL;

ID	HEALTHNUMBER	POLICYHOLDER	POLICYNUMBER	WORKERNAME
669a7f8a64aa46f2be6c1aaeafff6ed8	C232ND832	Critical Software	MH014343	António

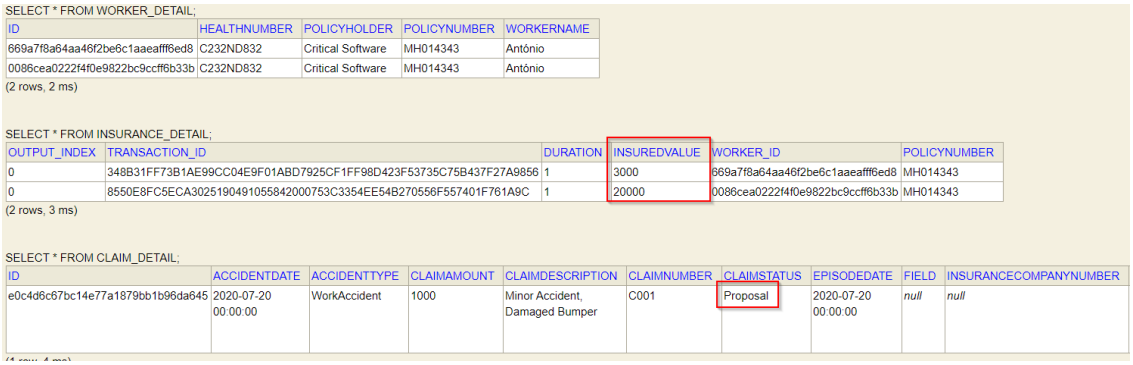
(1 row, 3 ms)

SELECT * FROM INSURANCE_DETAIL;

OUTPUT_INDEX	TRANSACTION_ID	DURATION	INSUREDVALUE	WORKER_ID	POLICYNUMBER
0	348B31FF73B1AE99CC04E9F01ABD7925CF1FF98D423F53735C75B437F27A9856	1	3000	669a7f8a64aa46f2be6c1aaeafff6ed8	MH014343

(1 row, 3 ms)

Figure 26. Worker Insurance Force Update



SELECT * FROM WORKER_DETAIL;

ID	HEALTHNUMBER	POLICYHOLDER	POLICYNUMBER	WORKERNAME
669a7f8a64aa46f2be6c1aaeafff6ed8	C232ND832	Critical Software	MH014343	António
0086cea0222f4f0e9822bc9ccff6b33b	C232ND832	Critical Software	MH014343	António

(2 rows, 2 ms)

SELECT * FROM INSURANCE_DETAIL;

OUTPUT_INDEX	TRANSACTION_ID	DURATION	INSUREDVALUE	WORKER_ID	POLICYNUMBER
0	348B31FF73B1AE99CC04E9F01ABD7925CF1FF98D423F53735C75B437F27A9856	1	3000	669a7f8a64aa46f2be6c1aaeafff6ed8	MH014343
0	8550E8FC5ECA3025190491055842000753C3354EE54B270556F557401F761A9C	1	20000	0086cea0222f4f0e9822bc9ccff6b33b	MH014343

(2 rows, 3 ms)

SELECT * FROM CLAIM_DETAIL;

ID	ACCIDENTDATE	ACCIDENTTYPE	CLAIMAMOUNT	CLAIMDESCRIPTION	CLAIMNUMBER	CLAIMSTATUS	EPISODEDATE	FIELD	INSURANCECOMPANYNUMBER
e0c4d6c67bc14e77a1879bb1b96da645	2020-07-20 00:00:00	WorkAccident	1000	Minor Accident, Damaged Bumper	C001	Proposal	2020-07-20 00:00:00	*null*	*null*

Figure 27. Claim Proposal Transaction

This “inconsistency” happens because Corda stores state data in three places. In NODE_TRANSACTIONS table, in VAULT_STATES table and, in the case of this PoC application, in three custom tables (WORKER_DETAIL, INSURANCE_DETAIL and CLAIM_DETAIL). The insurance value was changed in the custom schema table, which is only used as an interface to human-readable fields. The VAULT_STATES table only contains reference data, which is used to get the actual blob that resides in the NODE_TRANSACTIONS table that stores the data in binary format.

From the recommended tests for a blockchain project, only the performance category was not explored. It is still necessary to perform load testing to all Corda nodes to check the performance of the application. Aside from the performance testing, all the recommended categories were tested and presented positive results.

5.2 Evaluation

To evaluate the goals of this proof-of-concept application, the performed tests, presented in section 5.1.1, were essential, since they demonstrate the technical quality through unit, integration and functional tests.

The privacy of the application, the transactions validity, and nodes auditability are concerns that were taken into consideration and are thoroughly tested with both failure and success cases.

The results were positive since it was possible to verify that the application nodes can communicate with each other without relying on a central authority to be the middleman. The decentralization of data provides higher security than a centralized solution because there is no

single point of failure since no machine holds all the data records in the network, delaying the possibility of successful attacks. The privacy concern was also verified and returned a positive result since transaction data is only shared with the participants of the transaction and not by the entire network. Additionally, the data security, regarding authorization, also returned a positive result, since the transaction is only approved if both participants signatures are present in it.

In conclusion, with this proof-of-concept, it is possible to reduce the responsibilities of the central authority distributing it among the interested parties and still improve the security, privacy, and availability.

6 Conclusions

This chapter describes the stages followed to accomplish the main goal as well as the difficulties encountered, and the results obtained with the work developed. Additionally, this chapter also presents the improvements that can still be done to this project.

6.1 Work Summary

This dissertation aims to bring value to the insurance sector by using the existing properties of a distributed ledger technology to develop a proof-of-concept application that will help insurance companies to protect the insurance data. This goal required in-depth knowledge of the existing distributed ledger technologies, their data structure, the transaction validation mechanism, and other properties such as maturity.

The initial phase of the project consisted of research and data collection with the view of choosing the distributed ledger technology and the appropriate platform. A meeting was scheduled with members of Critical Software, including one of the developers of the existing APS solution, the Chief Technology Officer, and the APS client manger. The purpose of the meeting was to obtain a demo in a test environment to gather the necessary information on the requirements of the use case, as well as to assess how the existing solution is working.

Following the meeting, an investigation was carried out to study the different distributed ledger technologies and decide which one was the most suitable for the business case in question. The results of this investigation led to the decision of a blockchain-based solution. The next step was to choose, from the platforms available in the market, the best-suited one for the business case. The choice proved to be challenging because it involved the consideration of several factors,

namely, the requirements provided at the meeting, the decision of using a blockchain solution, and the relation between the two. After considering each one of these factors, the Corda open-source blockchain platform was chosen to develop the proof-of-concept application.

The development of the Corda application is the last phase of this project. It shows how the chosen distributed ledger technology can be applied to the insurance market, and most specifically, how it can help share data related to Workers Compensation Insurance without the need of an intermediate and without compromising the security and privacy of the system.

6.2 Contributions

This dissertation was initially intended for APS' use, and APS was meant to be part of the project and act as a stakeholder, providing feedback and evaluating the solution. Unfortunately, that was not possible, and the project became only for Critical Software's use. The contact with APS will possibly be resumed in the future. Despite this adjustment, the main goal was successfully met. A private distributed ledger application can bring benefits to the insurance sector by:

- Reducing the responsibilities of the central authority. Data is not owned by the central authority but by all nodes in the network.
- Improving data Security. Having data decentralized, there is not a single point of failure, and the system is more resilient to attacks.

The work behind this dissertation is highly relevant to both academic and business environments since it facilitates the study of distributed ledger technologies and their benefits to the insurance sector. This dissertation provides the possibility to assess, for a specific ledger, whether insurance data is being shared in a protected way, without compromising security and privacy.

The study of the best DLT platform for the insurance sector can also be used to investigate further how platforms are complying with insurance business requirements.

This dissertation can also be used to facilitate further studies of other Corda applications, including the trade-off between security and privacy that is already being addressed in a white paper entitled "Solutions for the Corda Security and Privacy Trade-off: Having Your Cake and Eating It", published by Dutch bank ING (Koens, 2019). The code developed for the CorDapp Workers Compensation Insurance is available in a public repository in GitHub:

https://github.com/CatarinaFranco94/workers-insurance.

As shown above, the business case and the positive contributions achieved were focused on the Workers Compensation Insurance (against accidents at work). However, at the beginning of this dissertation process, the work's scope and the purpose of the distributed ledger technology were not well defined. There was uncertainty as to the type of insurance, and therefore the research was very comprehensive and included other types of insurances, in addition to insurance against accidents at work. This research and the opportunity areas identified in the insurance value-chain can be revisited and further explored in the future as can be seen in Attachment A. Although the code needs to be adapted, this dissertation's work can serve as the basis for the development of other application which can be used for other types of insurance.

6.3 Next Steps

As for future work, this proof-of-concept application code can be improved, adding some new functionalities such as:

1. Data streams to share attachments in RPC calls. These attachments allow sharing PDF documents, such as scans of the medical exams performed to the worker.

   There are some aspects to have in consideration using attachments since it can overcome the size of a transaction. In this case, cloud storage can be a possibility to leave the data off-chain.

2. Add new data to the rejection flow, such as the reason for the denial.
3. Add a new condition to the insurance contract to verify that the claim being presented does not overcome the insurance value in the policy.
4. Add a "forget me" flow to clear all the worker data once they leave the company.
5. Add an invoice flow to inform the insurance company about the amount that was paid by the worker.

The improvements can also be made in terms of non-functional requirements such as:

6. Integrate the application logs with elastic search and Kibana to keep up with everything that is happening.
7. Monitor the application nodes using Prometheus and its alerts.

8. Deploy nodes databases on a cloud system to improve the availability of the application.
9. Deploy the Spring Boot Web Server in Docker.

References

Adviser, U. G. C. S. (2016). *Distributed Ledger Technology: beyond block chain*. Retrieved from https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/492972/gs-16-1-distributed-ledger-technology.pdf.

Androulaki, E., Barger, A., Bortnikov, V., Cachin, C., Christidis, K., De Caro, A., . . . Yellick, J. (2018). Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains. *arXiv e-prints*, arXiv:1801.10228. https://ui.adsabs.harvard.edu/abs/2018arXiv180110228A

Anwar, H. (2018). Comparison of Different Types of Distributed Ledgers: Blockchain vs Hashgraph vs Dag vs Holochain. Retrieved from https://101blockchains.com/blockchain-vs-hashgraph-vs-dag-vs-holochain/

Anwar, H. (2019). Distributed Ledger Technology. Retrieved from https://101blockchains.com/distributed-ledger-technology-dlt/

B3i. About Us. Retrieved from https://b3i.tech/who-we-are.html

Baird, L. (2016). The swirlds hashgraph consensus algorithm: Fair, fast, byzantine fault tolerance. *Swirlds, Inc. Technical Report SWIRLDS-TR-2016, 1*, 28.

Bashir, I. (2018). *Mastering Blockchain: Distributed ledger technology, decentralization, and smart contracts explained, 2nd Edition* (2 ed.): Packt Publishing.

Beers, B. (2015). Brief Overview of the Insurance Sector.

Benčić, F. M., & Žarko, I. P. (2018, 2-6 July 2018). *Distributed Ledger Technology: Blockchain Compared to Directed Acyclic Graph.* Paper presented at the 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS).

BitShares. (2014). Delegated Proof-of-Stake Consensus. Retrieved from https://how.bitshares.works/en/master/technology/what_bitshares.html#consensus-technology

Blockchain, C. (2015). About us. Retrieved from https://www.cambridge-blockchain.com/about

Board, F. S. (2019). Monitoring of FinTech. Retrieved from https://www.fsb.org/work-of-the-fsb/policy-development/additional-policy-areas/monitoring-of-fintech/

Brenchley, P. (2018). Blockchain in insurance. Retrieved from https://home.kpmg/xx/en/home/insights/2018/09/blockchain-in-insurance-fs.html

Brown, R. G. (2017). What slack can teach us about privacy in enterprise blockchains. Retrieved from https://gendal.me/tag/fabric/

Brown, R. G. (2019). *Corda Platform White Paper*. Retrieved from R3: https://www.r3.com/wp-content/uploads/2019/06/corda-platform-whitepaper.pdf

Burkhardt, D. (2018). *Distributed Ledger: Definition & Demarcation*.

Burrows, M. (2006). *The Chubby lock service for loosely-coupled distributed systems*. Paper presented at the Proceedings of the 7th symposium on Operating systems design and implementation, Seattle, Washington. https://dl.acm.org/doi/10.5555/1298455.1298487

Carson, B. (2018). Blockchain beyond the hype: What is the strategic business value? *Mc Kinsey Digital*.

Castro, M., & Liskov, B. (1999). *Practical Byzantine fault tolerance*. Paper presented at the Proceedings of the third symposium on Operating systems design and implementation, New Orleans, Louisiana, USA.

Chung, L., Nixon, B., Yu, E., & Mylopoulos, J. (2000). *Non-Functional Requirements in Software Engineering* (Vol. 5).

Churyumov, A. (2016). *Byteball: A Decentralized System for Storage and Transfer of Value*. Retrieved from Obyte: https://obyte.org/Byteball.pdf

Consensys. (2020). Blockchain in Insurance. Retrieved from https://consensys.net/blockchain-use-cases/finance/insurance/

Cooper, C. (2018). B3i selects Corda blockchain platform [Press release]. Retrieved from https://b3i.tech/news-reader/b3i-selects-corda-blockchain-platform.html

Corda. Contracts. Retrieved from https://docs.corda.net/docs/corda-os/4.5/key-concepts-contracts.html

Corda. Flows. Retrieved from https://docs.corda.net/docs/corda-os/4.5/key-concepts-flows.html

Corda. Introducing Corda, Discover the Corda platform. *Training Corda.* Retrieved from https://training.corda.net/key-concepts/introduction/

Corda. Key Concepts. https://docs.corda.net/docs/corda-os/4.5/key-concepts-states.html

Corda. Nodes. Retrieved from https://docs.corda.net/docs/corda-os/4.5/key-concepts-node.html

Corda. Transactions. Retrieved from https://docs.corda.net/docs/corda-os/4.5/key-concepts-transactions.html

Corda (Producer). (2018a, 2020-08-16). Corda Bootcamp 15 - Flows Theory. *Corda Bootcamp*. Retrieved from https://www.youtube.com/watch?v=FzXTiUvbCTQ

Corda. (2018b). High Availability for R3 Corda Nodes. Retrieved from https://www.corda.net/blog/high-availability-for-r3-corda-nodes/

Corda. (2019a). Advantages. Retrieved from https://www.corda.net/advantage/

Corda. (2019b). Corda v Hyperledger v Quorum v Ethereum v Bitcoin. Retrieved from https://www.corda.net/blog/corda-v-hyperledger-v-quorum-v-ethereum-v-bitcoin/

Corda, R. (2019). Corda Platform. Retrieved from https://www.r3.com/corda-platform/

Curtis, H. (2019). Axa stops selling blockchain flight delay product. *Insurance Post*. Retrieved from https://www.postonline.co.uk/personal/4461796/axa-stops-selling-blockchain-flight-delay-product

El Ioini, N., & Pahl, C. (2018). A Review of Distributed Ledger Technologies. In (pp. 277-288).

Fabric, H. (2020). *Hyperleder Fabric White Paper*. Retrieved from https://www.hyperledger.org/wp-content/uploads/2020/03/hyperledger_fabric_whitepaper.pdf

Fan, K., Wang, S., Ren, Y., Li, H., & Yang, Y. (2018). MedBlock: Efficient and Secure Medical Data Sharing Via Blockchain. *Journal of Medical Systems, 42*(8), 136. doi:10.1007/s10916-018-0993-7

Farshidi, S., Jansen, S., España, S., & Verkleij, J. (2020). Decision Support for Blockchain Platform Selection: Three Industry Case Studies. *IEEE Transactions on Engineering Management*, 1-20. doi:10.1109/TEM.2019.2956897

FBI. (2010). Insurance Fraud. Retrieved from https://www.fbi.gov/stats-services/publications/insurance-fraud

Finance, T. G. I. L. f. C. (2019). Blockchain Climate Risk Crop Insurance. Retrieved from https://www.climatefinancelab.org/project/climate-risk-crop-insurance/

Forum, C. (2019). *Insurance and Distributed Ledger Technology*. Retrieved from The CRO Forum: https://www.thecroforum.org/wp-content/uploads/2019/06/CROF_Insurance-and-DLT_2019-06-18-Final-1.pdf

Forum, W. E. (2016). *The future of financial infrastructure*. http://www3.weforum.org/docs/WEF_The_future_of_financial_infrastructure.pdf

Gaia, C. d. R. P. d. (2005). *Acidentes de Trabalho e Doenças Profissionais em Portugal*. Retrieved from http://www.crpg.pt/investigacao/Documents/coleccao/Estudos_CRPG_n5.pdf

Hashgraph, H. (2019). What is gossip about gossip. Retrieved from https://www.hedera.com/learning/what-is-gossip-about-gossip

Hays, D. (2018). Blockchain 3.0 The Future of DLT? *Coin Corner*. https://cryptoresearch.report/crypto-research/blockchain-3-0-future-dlt/

Hearn, M., & Brown, R. G. (2019). Corda: A distributed ledger. https://www.r3.com/wp-content/uploads/2019/08/corda-technical-whitepaper-August-29-2019.pdf

Holbrook, J. (2020). *Architecting Enterprise Blockchain Solutions*: Wiley.

Hoxha, L. (2018). *Hashgraph the Future of Decentralized Technology and the End of Blockchain*.

Hyperledger. (2018). *Introdution to Hyperledger Whitepaper*. Retrieved from https://www.hyperledger.org/wp-content/uploads/2018/07/HL_Whitepaper_IntroductiontoHyperledger.pdf

IBM. (2019). Marsh - Transforming proof of insurance with blockchain. Retrieved from https://www.ibm.com/case-studies/marsh-ibm-blockchain

Ifegwu, O. (2019). Mainnet. *Glossary.* Retrieved from https://academy.binance.com/glossary/mainnet

Insights, C. (2019). How Blockchain Could Disrupt Insurance.

Insights, L. (2019). USAA partners with State Farm for auto blockchain insurance solution. *Insurance*. Retrieved from https://www.ledgerinsights.com/usaa-state-farm-blockchain-insurance-solution-subrogation/

(2018). *Fizzy, AXA's Blockchain Case Study*

(2019). *State Farm's blockchain – Auto Claims Subrogation*. Retrieved from https://insureblocks.com/ep-69-state-farms-blockchain-auto-claims-subrogation/

Insureon. (2020). Subrogation. Retrieved from https://www.insureon.com/insurance-glossary/subrogation

Insurwave. (2019). Insurwave Blockchain-enabled Marine Insurance. Retrieved from https://insurwave.com/

Jelurida. (2013, 2020-05-11). What is Nxt? Retrieved from https://www.jelurida.com/nxt/what-is-nxt

Koen, P. (2007). The Fuzzy Front End for Incremental, Platform, and Breakthrough Products. In *The PDMA Handbook of New Product Development* (2 ed., pp. 81-91).

Koen, P., Ajamian, G., Burkart, R., Clamen, A., Davidson, J., D'Amore, R., . . . Wagner, K. (2001). Providing Clarity and A Common Language to the "Fuzzy Front End". *Research-Technology Management, 44*(2), 46-55. doi:10.1080/08956308.2001.11671418

Koens, T., King, S., Bos, M. v. d., Wijk, C. v., & Koren, A. (2019). Solutions for the Corda Security and Privacy
Trade-off: Having Your Cake and Eating It.
https://mondovisione.com/_assets/files/Corda_DoSt_v1.6.pdf

Kudu, A. (2019). Introducing Apache Kudu. Retrieved from https://kudu.apache.org/docs/

KYC-Chain. (2019a). About us. Retrieved from https://kyc-chain.com/about-us/

KYC-Chain. (2019b). KYC-Chain partners with leading Incorporations platform, Korporatio. Retrieved from https://kyc-chain.com/kyc-chain-partners-with-korporatio/

Lai, R., & Lee Kuo Chuen, D. (2018). Chapter 7 - Blockchain – From Public to Private. In D. Lee Kuo Chuen & R. Deng (Eds.), *Handbook of Blockchain, Digital Finance, and Inclusion, Volume 2* (pp. 145-177): Academic Press.

Lake, S. (2016). Proof of Elapsed Time (PoET). Retrieved from https://sawtooth.hyperledger.org/docs/core/releases/0.7/introduction.html#consensus

Lamport, L. (1998). The part-time parliament. *ACM Trans. Comput. Syst., 16*(2), 133–169. doi:10.1145/279227.279229

Lang, K. (2016). *Kotlin Language Documentation*. Retrieved from https://kotlinlang.org/docs/kotlin-docs.pdf

Lehman, M. (2019). Blockchain in insurance: from experimentation to real-world value. Retrieved from https://insuranceblog.accenture.com/blockchain-in-insurance-from-experimentation-to-real-world-value

Liu, M., Wu, K., & Xu, J. (2019). How Will Blockchain Technology Impact Auditing and Accounting? Permissionless vs. Permissioned Blockchain. *Auditing eJournal, 13*. doi:10.2308/ciia-52540

Lu, Y. (2019). The Blockchain: State-of-the-Art and Research Challenges. *Journal of Industrial Information Integration, 15*. doi:10.1016/j.jii.2019.04.002

Marsh. (2019). Marsh to Begin Rollout of Proof of Insurance Blockchain Platform. *Marsh*. Retrieved from https://www.marsh.com/us/media/proof-of-insurance-blockchain-platform.html

McDonald, T. (2018). Introducing Insurwave, built on the Corda blockchain. Retrieved from https://www.r3.com/blog/introducing-insurwave-built-on-the-corda-blockchain/

Morgan, J. P. About us. Retrieved from https://about.jpmorganchase.com/about

Nakamoto, S. (2008). Bitcoin: A Peer-to-Peer Electronic Cash System. *Cryptography Mailing list at https://metzdowd.com*.

News, C. (2018). Leading Indian Life Insurers Partner with Cognizant to Develop Industry-Wide Blockchain Solution for Secure Data-Sharing and Improved Customer Experience [Press release]. Retrieved from https://news.cognizant.com/2018-04-16-Leading-Indian-Life-Insurers-Partner-with-Cognizant-to-Develop-Industry-Wide-Blockchain-Solution-for-Secure-Data-Sharing-and-Improved-Customer-Experience-1

Nicoletti, B. (2017). *The Future of FinTech: Integrating Finance and Technology in Financial Services* (1st ed.): Springer International Publishing.

Obyte. (2016). Obyte News [Press release]. Retrieved from https://wiki.obyte.org/News

Ongaro, D., & Ousterhout, J. (2014). *In search of an understandable consensus algorithm*. Paper presented at the Proceedings of the 2014 USENIX conference on USENIX Annual Technical Conference, Philadelphia, PA.

Osterwalder, A., Pigneur, Y., & Clark, T. (2010). *Business Model Generation: A Handbook for Visionaries, Game Changers, and Challengers*: Wiley.

Peercoin. (2012). Fair Distribution. Retrieved from https://www.peercoin.net/

Peffers, K., Tuunanen, T., Rothenberger, M. A., & Chatterjee, S. (2007). A Design Science Research Methodology for Information Systems Research. *Journal of Management Information Systems, 24*(3), 45-77. doi:10.2753/MIS0742-1222240302

Persistent. (2018). Blockchain Based eKYC (Know Your Customer). Retrieved from https://www.persistent.com/new-and-emerging-tech/blockchain/know-your-customer/

Planeamento, G. d. E. e. (2013). Acidentes de Trabalho. Retrieved from http://www.gep.mtsss.gov.pt/acidentes-de-trabalho

Popov, S., Saa, O., & Finardi, P. (2019). Equilibria in the tangle. *Computers & Industrial Engineering, 136*, 160-172. doi:https://doi.org/10.1016/j.cie.2019.07.025

Portugal, C. E. d. (2018). Associação Portuguesa de Seguradores. Retrieved from https://cip.org.pt/aps-associacao-portuguesa-de-seguradores/

Purpura, P. (2013). Resilience, Risk Management, Business Continuity, and Emergency Management. In (pp. 321-362).

QA, M. B. (2018). Testing Blockchain applications. Retrieved from https://www.magicblockchainqa.com/our-services/

Research, J. (2017). IBM Ranked No 1 Blockchain Technology Leader [Press release]. Retrieved from https://www.juniperresearch.com/press/press-releases/ibm-ranked-no-1-blockchain-technology-leader

Risk, T. I., & Group, I. K. (2017). *Blockchain Building Blocks: Creating a world of opportunity for insurance from an evolving area of technology*. Retrieved from https://www.theinstitutes.org/doc/rba-alliance/Blockchain_Building_Blocks.pdf

Rizzo, P. (2016). Insurance Giant MetLife Joins R3 Blockchain Consortium [Press release]. Retrieved from https://www.coindesk.com/insurance-giant-metlife-joins-r3-blockchain-consortium

(2019). *Ep. 80 – B3i's new CEO & product launch* [Retrieved from https://insureblocks.com/ep-80-b3is-new-ceo-product-launch/

Schmid, P. (2019). *Examining The Benefits of The Institutes RiskStream Collaborative's Proof of Insurance and First Notice of Loss Applications*. Retrieved from The Institutes: https://www.theinstitutes.org/doc/rba-alliance/Blockchain_Building_Blocks.pdf

Services, A. A. o. I. Introducing openIDL. Retrieved from https://aaisonline.com/aais-openidl

Services, A. A. o. I. Our Role in Insurance. Retrieved from https://aaisonline.com/our-role-in-insurance

Services, A. A. o. I. (2019). AAIS Solutions Kit - OpenIDL. Retrieved from https://www.aaisonline.com/documents/20143/2186560/openIDL-Fact+Sheet.pdf/69adb41e-40bb-9072-b87a-2ed3bff76352?t=1534874499348

Spadoni, A. (2018). Doing business in a Blockchain native world. Retrieved from https://b3i.tech/news-reader/doing-business-in-a-blockchain-native-world.html

Supervisors, I. A. o. I. (2017). *FinTech Developments*. Retrieved from https://www.iaisweb.org/file/65625/report-on-fintech-developments-in-the-insurance-industry.pdf

Takyar, A. (2018). Blockchain vs Hedera Hashgraph. https://www.leewayhertz.com/hashgraph-vs-blockchain/

Tendermint. (2014). What is Tendermint? Retrieved from https://docs.tendermint.com/master/introduction/what-is-tendermint.html

Thomas, H., Matthew, S., Philip, E., Alexander, S., & Bela, G. (2018). On-chain vs. off-chain storage for supply- and blockchain integration. *it - Information Technology, 60*(5-6), 283-291. doi:https://doi.org/10.1515/itit-2018-0019

Thompson, E. (2019). Hashgraph vs blockchain: The differences you need to know. Retrieved from https://coinrivet.com/hashgraph-vs-blockchain-the-differences-you-need-to-know/

Tocha, C. (2018). Workplace accident insurance. Retrieved from https://www.e-konomista.pt/seguro-de-acidentes-de-trabalho/

Verbeeten, D., & McGrath, S. (Writers). (2019). Insurance With, On and For Blockchain. In: ConsenSysMedia.

Vikram. (2019). Blockchain vs Hashgraph: A thorough analysis. https://www.mobileappdaily.com/hashgraph-vs-blockchain/amp

Wales, I. o. C. A. i. E. a. (2019). History of blockchain. Retrieved from https://www.icaew.com/technical/technology/blockchain/blockchain-articles/what-is-blockchain/history

Xia, Q., Sifah, E. B., Asamoah, K. O., Gao, J., Du, X., & Guizani, M. (2017). MeDShare: Trust-Less Medical Data Sharing Among Cloud Service Providers via Blockchain. *IEEE Access, 5*, 14757-14767. doi:10.1109/ACCESS.2017.2730843

Xu, X., Weber, I., Staples, M., Zhu, L., Bosch, J., Bass, L., . . . Rimba, P. (2017, 3-7 April 2017). *A Taxonomy of Blockchain-Based Systems for Architecture Design.* Paper presented at the 2017 IEEE International Conference on Software Architecture (ICSA).

Zaman, Z. (2019). *How MetLife Applied Blockchain to Solve a Difficult Health Insurance Challenge*. Paper presented at the From Vision to Reality. https://www.cognizant.com/whitepapers/from-vision-to-reality-how-metlife-applied-blockchain-codex4398.pdf

Zeithaml, V. (1988). Consumer Perceptions of Price, Quality and Value: A Means-End Model and Synthesis of Evidence. *Journal of Marketing, 52*, 2-22. doi:10.1177/002224298805200302

Zheng, Z., Xie, S., Dai, H., Chen, X., & Wang, H. (2017, 25-30 June 2017). *An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends.* Paper presented at the 2017 IEEE International Congress on Big Data (BigData Congress).

Attachment A – Promising areas of opportunity in the insurance value chain

The potential advantages of a distributed ledger technology for the insurance industry has gained attention since the early years of Blockchain technology. Blockchain-based solutions are already transforming the industry with more than 700 start-ups spread around the world, fulfilling the most promising areas of opportunity in the insurance value chain (Table 9).

Table 9. Opportunity Areas in the Insurance Value Chain

Products, Pricing and Distribution	**Underwriting and Risk Management**	**Policyholder Acquisition and Servicing**	**Claims Management**	**Finance, Payments and Accounts**	**Regulation and Compliance**
Parametric insurance	Data sharing and risk registries	Policyholder acquisition	Fraud Register	Netting and payments across countries	Real-time regulatory monitoring
Mobile insurance for developing countries	Peer-to-peer insurance	Know Your Customer (KYC) / Anti-Money Laundering (AML) efforts	Claims Automation	Subrogation	Proof of insurance
Insurance included in transactional purchases	Provenance	Placement documentation	Multi-layer claims settlement	New forms of raising capital (crowd-founding)	Education / licensing catalogue

Blockchain can benefit both insurers and insureds. From an insurer's point of view, the usage of blockchain technology can lower costs, ease data management, simplify processes, offer new products, combat fraud and lower regulatory burdens. From an insured's perspective, the usage

of blockchain technology can enhance the customer experience, improve affordability, provide a means for more significant product innovation and allow for faster entry into emerging markets (Schmid, 2019).

The next sections introduce the areas of opportunity in the insurance value chain. For each area, some use cases, that have already been explored by the blockchain technology, are presented and analysed. In some cases, the final product is already in production, and those cases are mention.

Products, pricing and distribution

Insurance products, such as parametric insurance and IoT-based products can increase the efficiency in the insurance process and therefore, lower prices and reach emerging markets (Risk & Group, 2017). The insurance industry is already exploring the most promising areas of opportunities in this section.

Parametric Insurance

Parametric insurance is an agreement between the insurer (entity assuming risk) and the insured (entity covered under an insurance policy). The insurer agrees to pay the indemnitee an agreed amount upon the occurrence of a specified event, such as an earthquake or hurricane of specified intensity. This type of insurance if simpler than the traditional one. This use case fits perfectly in a blockchain project, and it is widely used for companies.

Fizzy, AXA's blockchain product for flight delay insurance policy, is a fully automated flight delay insurance policy that runs on the Ethereum public blockchain and allows customers to get indemnified as soon as they arrive at their destination. The process is fully automated, triggered by an Ethereum smart contract that is connected to a plane arrival time aggregator deciding whether customers are eligible for indemnification. This automation means no action is required by eligible customers to claim their indemnity (Insureblocks, 2018). Fizzy development started in late 2015 and was launched in September 2017, starting with just a few routes from Paris Charles de Gaulle airport to the US before expanding to Italy (Insureblocks, 2018). Potential partners include online travel agencies, airports and travel apps.

Fizzy has been discontinued after two years (in November 2019). AXA has struggled to reach commercial targets with its first Fizzy product, claiming there is not enough market appetite for

a blockchain-based consumer insurance product yet (Curtis, 2019). Fizzy was still a pioneer of sorts and has laid the groundwork for future blockchain insurance platforms.

Mobile insurance for developing countries

Developing countries mainly include those nations which are relatively and similarly weak socially and economically. While the potential to progress, these nations, unfortunately, do not have adequate access to present-day technology, primarily due to lack of infrastructure. These nations could use transparency, security, and accountability in their processes, all of which are cornerstones of Blockchain technology [Developing Countries and Blockchain Technology: Uganda's Perspective].

Concerning developing countries, blockchain technology can step in and provide a way around existing rules and regulations [Developing Countries and Blockchain Technology: Uganda's Perspective].

Blockchain Climate Risk Crop Insurance is a platform where insurance policies are plugged into blockchain smart contracts and indexed to local weather. In the eventuality of an extreme weather event, the policies are automatically triggered, which facilitates fair, transparent and timely pay-outs.

For farmers, this solution can reduce transaction costs during the processing of claims. It is estimated that in the long-term, the cost required to issue policy can reduce by 41% (Finance, 2019).

The pilot will take between two to four years, starting from April 2020, and will aim to insure 1.2 million farmers in Kenya, who will be able to pay the insurance premiums in small instalments as low as 50 cents (Finance, 2019).

Underwriting and risk management

Distributed ledger technology could also be beneficial to underwriting and risk management. The immutability associated with blockchain data can provide trustworthy audibility features (Risk & Group, 2017).

Data sharing and risk registries

One of the most promising implementations of blockchain is around sharing data – enhancing existing business practices through better data and data-sharing (Verbeeten & McGrath, 2019).

Data sharing is, exactly, what this dissertation is trying to improve – sharing information, about working accidents insurance claims, between multiple parties. Claims are recorded in a shared database, enhancing the quality of data and achieving efficiency (Consensys, 2020):

- Creating an immutable and trustworthy record for the benefit of all stakeholders;
- Enhance industry-wide efforts to mitigate claims fraud by superior data sharing;
- Track claims in real-time and even across borders.

Blockchain technology can enable better coordination between stakeholders. On a distributed ledger, insurers can record permanent transactions, with access controls to protect data security and identify suspicious behaviour across the ecosystem (C. Insights, 2019).

The technology can be utilised as a method to seamlessly and securely share data among decentralized institutions, and may also minimise counterfeiting, double booking, and document or contract alterations by establishing clear, timeless records of asset ownership.

Cognizant is a services companies that transform clients' business, operating and technology models for the digital era.

Cognizant, alongside with a consortium of leading Indian life insurers, is prototyping a new blockchain solution for **insurance data sharing** built on Corda, R3's distributed ledger platform and hosted on Microsoft Azure.

The solution ensures real-time availability, consistency and transparency, reducing the risk of data leaks and double-spend, delivering an improved experience to the customer (News, 2018). Additionally, the availability and the traceability of store data, blockchain can reduce operating costs, avoiding duplication of procedures and streamline approvals (News, 2018).

Policyholder Acquisition and Servicing

Policyholder acquisition and servicing can become more efficient and improve the customer experience - insurers do not have to collect all the client information manually since the insurance information is already securely stored in the blockchain. Besides, insurance life cycle

documents will be easily tracked, avoiding duplicated registers and verification concerns about know-your-customer regulations (Risk & Group, 2017).

Know Your Customer (KYC) / Anti-Money Laundering (AML) efforts

Each financial institution has to indulge in the Know-Your-Customer (KYC) process with customer to comply with regulations, such as Anti-Money-Laundering (AML). Each financial institution performs its customer checks. The customer typically provides KYC documents each time he requires services within the same institution. Institutions are unable to easily share customer KYC information securely and quickly while retaining the confidentiality and privacy of customer documents. This constant verification of KYC documents results in poor customer experience and high operational costs for financial institutions (Persistent, 2018).

Blockchain technology allows financial institutions to securely share and exchange KYC information with a clear audit trail generated on the blockchain.

KYC-Chain is a compliance dashboard and Whitelabel customer onboarding portal, enabling companies to perform due diligence on their customers under AML/KYC requirements. This product, built on Ethereum, is already available for trail use (KYC-Chain, 2019a, 2019b).

Cambridge Blockchain returns control of personal data to users while delivering the benefits of a trusted distributed identity to consumers and organizations. Founded in 2015, Cambridge Blockchain has grown across three continents, in the data privacy, technology and business (Blockchain, 2015).

Claims Management

When an insured party incurs a loss, a claim is made to the insurer to cover the damage as stipulated in the insurance contract. For an insurance company, the settling of losses and adjusting differences between itself and the policyholder is known as claims management (Purpura, 2013).

Claims management could be simplified through smart contracts - which can ensure that claims are managed in a transparent, efficient and irrefutable manner - while an industry-wide shared ledger could help with multilayer settlements and fraud inspection (Risk & Group, 2017).

Fraud Register

Fraud losses represent an increasingly complex and systemic risk that's affecting the profitability of every insurance company. In the US, the total cost of insurance fraud (excluding health insurance) is estimated to be more than $40 billion per year (FBI, 2010).

Distributed ledger technology can provide better coordination between ledger participants to combat fraud. The insurers could collaborate and identify suspicious behaviour in the transactions data (C. Insights, 2019).

OpenIDL (open Insurance Data Link) is a public blockchain network that streamlines regulatory reporting and provides new insights for insurers while enhancing timeliness, accuracy, and value for regulators (Services, 2019[Fact sheet]).

The American Association of Insurance Services (AAIS) is a national insurance advisory not-for-profit organization in the United States governed by its Member companies (Services). OpenIDL, governed by AAIS, is the first open blockchain platform that enables the efficient, secure, and permission-based collection and sharing of statistical data (policy, premium, claims and loss experience data) (Services, 2019).

AAIS serves as openIDL administrator, providing unbiased governance for the blockchain platform within existing legal and regulatory frameworks. OpenIDL works with IBM Blockchain to radically transform insurance operations with faster verifiable data exchanges, a foundation of transparency, and transactions underpinned with pervasive security and trust. IBM is recognised as the leading enterprise blockchain provider (Research, 2017), working with hundreds of clients across financial services, supply chain, government, retail, media, and healthcare to implement blockchain applications, and operates several networks running live and in production (Services, 2019).

OpenIDL runs on the IBM Blockchain Platform powered by the Hyperledger technology, which currently hosts 250 active blockchain networks and hundreds of individual blockchain projects across various industries. Hyperledger is an open-source collaborative effort created to advance cross-industry blockchain technologies hosted by the Linux Foundation (Services, 2019).

OpenIDL blockchain has a strict policy and, to participate in the ledger, it is necessary to be a member of AAIS through particular Data Affiliation and Membership agreements (Services).

Finance, Payments and Accounts

Finance, payments, and accounting could also improve using this new disruptive technology. A distributed ledger like a blockchain can lower international payments and use smart contracts to increase efficiency in the subrogation process (Risk & Group, 2017).

Subrogation

Subrogation is the replacement of one person or group for another in a legal setting. In the field of insurance, subrogation is when the insurance company stands in for the insured and assumes legal right to pursue an individual or organization for an insurance claim.

Subrogation is a technique used by insurance companies to reclaim the money paid out for insurance claims (Insureon, 2020).

State Farm, the largest U.S. property/casualty insurer and the United Services Automobile Association (USAA) are working together since 2018 in a blockchain solution for **auto insurance claims subrogation**.

Currently, the solution is being tested with real claims data to see how blockchain can speed up the subrogation process (L. Insights, 2019).

State Farm's blockchain solution built on a Quorum blockchain is governed by JP Morgan (leader of global financial services and one of the largest banking institutions in the United States, with operations worldwide (Morgan)).

Based on Ethereum, Quorum is an open-source blockchain platform that combines the innovation of the public Ethereum community with enhancements to support enterprise needs (Insureblocks, 2019).

Regulation and Compliance

Distributed ledger technology can transform insurance regulation, and compliance by monitor all insurance variables in real-time and potentially create proof of insurance ledger (Risk & Group, 2017). Real-time regulatory monitoring allows insurance companies to use up-to-date data to detect risk factors and provide a specific type of insurance for every customer.

Attachment B – Innovation process

Koen (Koen, 2007) described the innovation process as composed of three phases: Fuzzy Front End (FFE), New Product Development (NPD), and the Commercialization phases, as indicated in Figure 28.

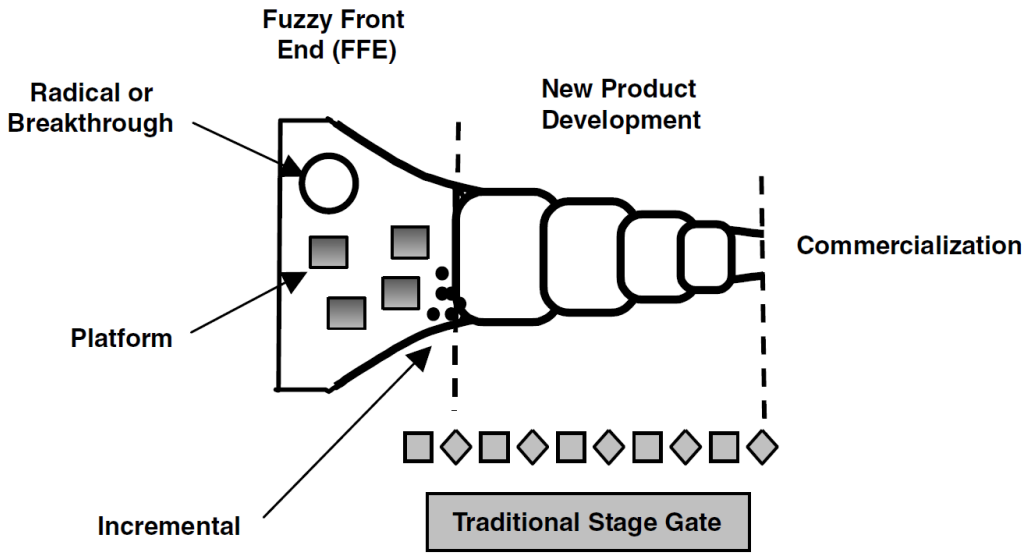


Figure 28. Parts of the innovation process

Source: (Koen, 2007)

- **Fuzzy Front End (FFE).** Provides the innovation concepts and business plans for the NPD phase. These pre-product development activities are often chaotic, unpredictable, and unstructured. The step generates innovative ideas and solutions which may be rigorously designed in the future steps.
- **New Product Development (NPD).** Is a set of formal, typically structured activities that cover the process of bringing a new product to market availability.
- **Commercialization.** The process of bringing the developed products to the market.

The front end of innovation, or what is often called the Fuzzy Front End, presents one of the most significant opportunities for improving the innovation process. This phase is receiving

more attention because it perceives that the New Process and Development phase was not receiving high-profit ideas coming from the FFE.

The **New Concept Development Model** (NCD), shown in Figure 29, provides a common language and definition of the FFE, and consists of three essential parts (Koen et al., 2001):

1. The inner area which defines the five elements of the Front End of Innovation (FEI).
2. The engine that drives the five front-end elements and is powered by the organization's leadership and culture.
3. The influencing factors consist of Organizational Capabilities, Business Strategy, and the Outside World – distribution, channels, customers, and competitors.

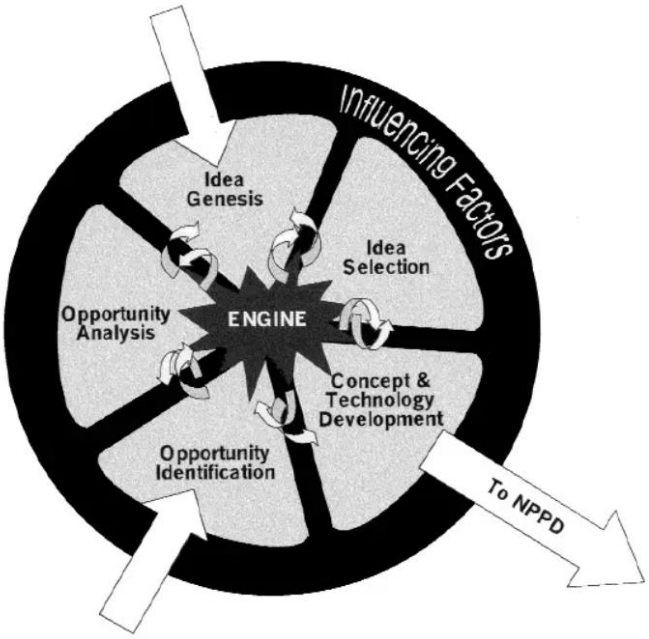


Figure 29. The New Concept Development Model (NCD)

Source: (Koen et al., 2001)

The front-end elements that are a part of the New Concept Development Model are (Koen et al., 2001):

1. Opportunity Identification. This is where the organization identifies the opportunities that might be interesting to pursue. Business and technological opportunities are considered, and resources allocated to new areas of market growth—the goals of the business drive this element.

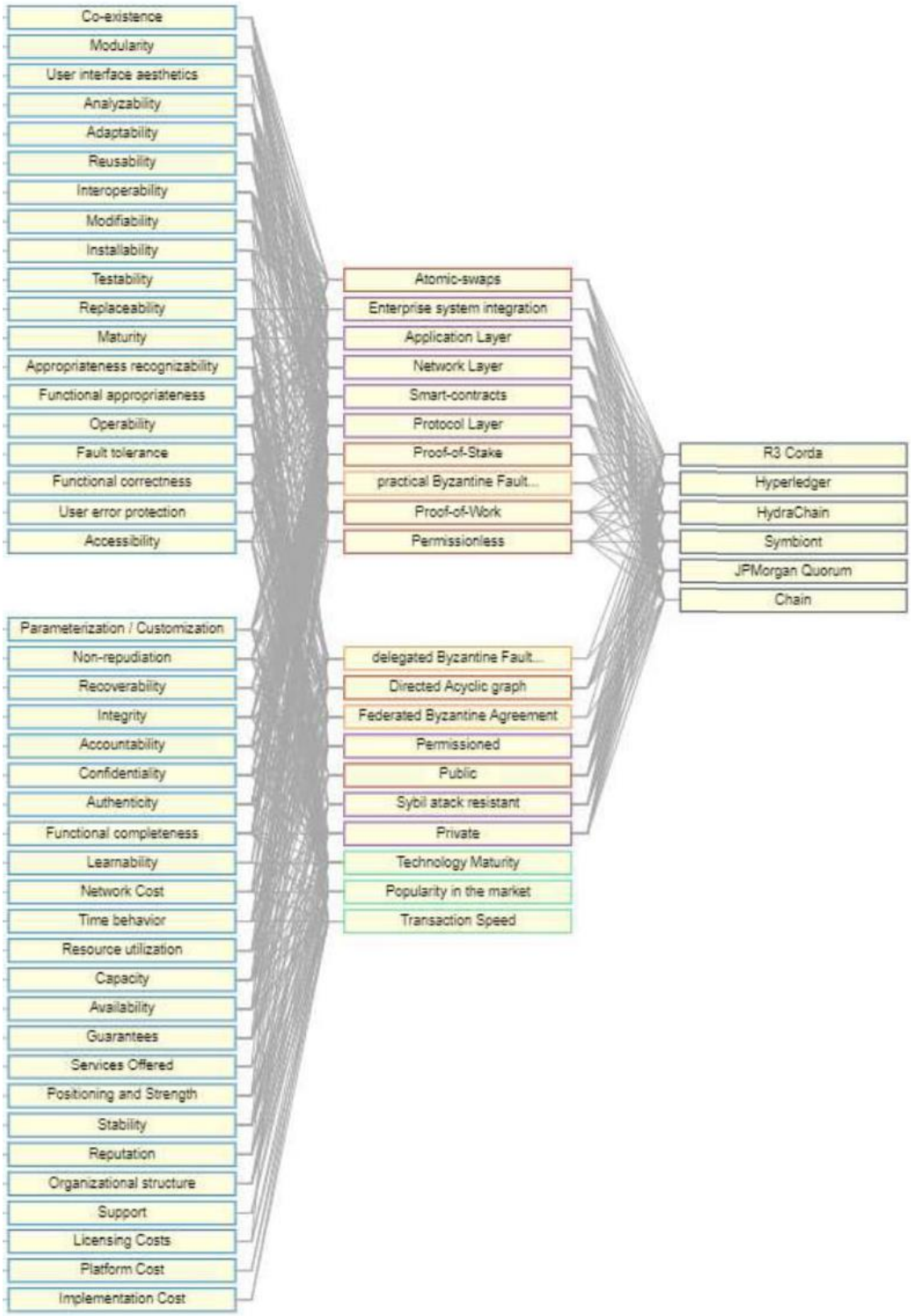
2. Opportunity Analysis. Translating an Opportunity Identification into specific business and technology opportunities requires additional information for making an early and uncertain technology and market assessments. Both competitive intelligence and trend analyses are used in this element.
3. Idea Genesis. Genesis is the birth, development, and maturation of the opportunity into a concrete idea. The idea may go through many iterations and changes as it is analysed, studied, discussed, and developed. The goal of this element is to generate new or modified ideas for the identified opportunity.
4. Idea Selection. This element goal is to choose which ideas achieve the most business value. It is important to note that idea selection must be less rigorous than opportunity analysis since the ideas should be allowed to grow and advance with less certainty.
5. Concept and Technology Development. The final element of the model involves the development of a business case based on estimates of market potential, customer needs, investment requirements technology unknows, and overall project risk.

The next step is to apply the five front-end elements, mention above, to this dissertation's context – Prototype to share insurance data using DLT:

1. Opportunity Identification. APS recognised that their responsibilities could be minimised with the use of a Distributed Ledger Technology.
2. Opportunity Analysis. APS recognised that DLT had shown an enormous potential which makes them rethink their business models.
3. Idea Genesis. APS has many business cases that could fit in the DLT context. For this idea genesis, the business case to be chosen has to be sufficiently capable of showing how efficient and secure DLT is, and, at the same time, has to be simple enough to be implemented thinking about the technology and not the complex business rules.
4. Idea Selection. The selected business context was the worker's compensation claim which was identified as one of the business cases that can most benefit from a DLT solution.
5. Concept and Technology Development. First, choose the best DLT for this context. And second, select the best platform, supported for the chosen DLT, and with the best characteristics for this context. This element output is the blockchain prototype, using Corda private blockchain, with the finality of sharing work insurance claims data

between different interlocutors and still guarantee the robustness and reliability of the data.

Attachment C – Blockchain Platform Decision Structure



The above figure illustrates part of the decision model created by the decision support system used to choose the best platform for workers insurance information sharing. In this figure is possible to see some of the DSS features that, according to the given prioritization, resulted in a list of six possible blockchain platforms.

Attachment D – Webserver postman collection


```
{
    "info": {
        "_postman_id": "cdf292d2-5c7d-4c20-b390-94ff63877804",
        "name": "Work Insurance",
        "schema":
"https://schema.getpostman.com/json/collection/v2.1.0/collection.json"
    },
    "item": [
        {
            "name": "Worker Insurance",
            "request": {
                "auth": {
                    "type": "noauth"
                },
                "method": "POST",
                "header": [
                    {
                        "key": "Content-Type",
                        "name": "Content-Type",
                        "type": "text",
                        "value": "application/json"
                    }
                ],
                "body": {
                    "mode": "raw",
                    "raw": "{\n\t\"workerInfo\": {\n\t\t\"policyNumber\":
\"MH014343\",\n\t\t\"healthNumber\": \"C232ND832\",\n\t\t\"workerName\":
\"António\",\n\t\t\"policyHolder\": \"Critical
Software\"\n\t},\n\t\"insuredValue\": \"20000\",\n\t\"duration\": 1\n}"
                },
                "url": {
                    "raw": "http://localhost:50005/workerInsurance/insuree-
HSJ",
                    "protocol": "http",
                    "host": [
                        "localhost"
                    ],
                    "port": "50005",
                    "path": [
                        "workerInsurance",
                        "insuree-HSJ"
                    ]
                }
            },
            "response": []
        },
        {
            "name": "Claim Acceptance",
            "request": {
                "method": "POST",
                "header": [
                    {
                        "key": "Content-Type",
                        "name": "Content-Type",
                        "value": "application/json",
                        "type": "text"
```

```
                    }
                ],
                "body": {
                    "mode": "raw",
                    "raw": "{\n\t\"insuranceCompanyNumber\":
\"12345DF\",\n\t\"insuranceCompanyPolicyNumber\": \"098776XO\",\n\t\"field\":
\"Acidente de trabalho\"\n}"
                },
                "url": {
                    "raw":
"http://localhost:50005/workerInsurance/acceptanceClaim/MH014343/C001/insuree-
HSJ",
                    "protocol": "http",
                    "host": [
                        "localhost"
                    ],
                    "port": "50005",
                    "path": [
                        "workerInsurance",
                        "acceptanceClaim",
                        "MH014343",
                        "C001",
                        "insuree-HSJ"
                    ]
                }
            },
            "response": []
        },
        {
            "name": "Claim Proposal",
            "request": {
                "method": "POST",
                "header": [
                    {
                        "key": "Content-Type",
                        "value": "application/json",
                        "type": "text"
                    }
                ],
                "body": {
                    "mode": "raw",
                    "raw":
"{\r\n\t\"claimNumber\":\"C001\",\r\n\t\"claimDescription\": \"Minor Accident,
Damaged Bumper\",\r\n\t\"claimAmount\":1000,\r\n\t\"internalPolicyNo\" :
\"123AB123\",\r\n\t\"accidentDate\" : \"2020-07-20\",\r\n\t\"episodeDate\" :
\"2020-07-20\",\r\n\t\"accidentType\" : \"WorkAccident\",\r\n\t\"module\" :
\"Urgency\"\r\n}"
                },
                "url": {
                    "raw":
"http://localhost:50007/workerInsurance/claim/MH014343",
                    "protocol": "http",
                    "host": [
                        "localhost"
                    ],
                    "port": "50007",
                    "path": [
                        "workerInsurance",
                        "claim",
                        "MH014343"
                    ]
                }
            },
            "response": []
        },
```

```
        {
            "name": "Claim Reject",
            "request": {
                "method": "POST",
                "header": [
                    {
                        "key": "Content-Transfer-Encoding",
                        "value": "application/json",
                        "type": "text"
                    }
                ],
                "url": {
                    "raw":
"http://localhost:50005/workerInsurance/rejectClaim//MH014343/C001/insuree-
HSJ",
                    "protocol": "http",
                    "host": [
                        "localhost"
                    ],
                    "port": "50005",
                    "path": [
                        "workerInsurance",
                        "rejectClaim",
                        "",
                        "MH014343",
                        "C001",
                        "insuree-HSJ"
                    ]
                }
            },
            "response": []
        }
    ],
    "event": [
        {
            "listen": "prerequest",
            "script": {
                "id": "b57b5042-a000-4b4a-8386-d777de42bd6a",
                "type": "text/javascript",
                "exec": [
                    ""
                ]
            }
        },
        {
            "listen": "test",
            "script": {
                "id": "c9f8043a-40e3-40be-8324-6c3e433f5935",
                "type": "text/javascript",
                "exec": [
                    ""
                ]
            }
        }
    ],
    "protocolProfileBehavior": {}
}
```