Blockchain Powered Platform for Consolidated, Shared and Trusted Healthcare

João Marques de Barros Mendes Leal

Mestrado em Engenharia Informática
Especialização em Engenharia de Software

Dissertação orientada por:
Prof. Doutor Pedro Manuel Barbosa Veiga

2019
Acknowledgments

I would like to start by thanking my dissertation supervisor Professor Doutor Pedro Veiga for all the help and support provided during this dissertation. Whether it was in our weekly e-mail exchanges, the meetings in his office to discuss the progress of the work or for all the help in writing and improving this report, I am very grateful for all of it.

To the host organization I am every thankful for giving me the opportunity to explore and develop a system in this very promising new technology. I would like to give a special acknowledgment to Tiago Minchin and Álvaro Silva for their indispensable assistance during the dissertation. For the invaluable knowledge of the industries considered and for always being available to provide the help needed to resolve every issue that came up during the project.

To all of the professors I had at the Faculty of Sciences during my bachelor’s and first year of the master’s I would to thank all of you for teaching me all the necessary knowledge and skill I needed to successfully perform the work for this dissertation.

A final special thank you is needed to all of my friends and family who always believed and supported me not only during this project but since I can remember.
To my family.
Abstract

In recent years blockchain, has become a topic of much debate and interest. This is due to the increase in valuation of cryptocurrencies and has attracted the attention of the public, corporations and governments. Cryptocurrencies use blockchain as the underlying technology. Blockchain refers to a ledger that keeps a permanent and immutable record of transactions and is replicated amongst all nodes. It offers guarantees of trust, immutability and transparency of the data, decentralization and elimination of third parties of data storage.

The healthcare industry is one of the biggest and most valuable industries in the world. It encompasses governments, public, private enterprises and citizens. However, there are some issues surrounding the current way the industry operates. Heavy decentralization, lack of communication between professionals and lack of trust in the information are just some of the issues that plague the industry.

Considering these facts, the purpose of this dissertation is to create a platform to study the applicability of blockchain to solve the problems of the healthcare industry. This platform will store medical data of patients, allow them to manage access to it and give medical professionals the capability to review, add or update the information of the person they are treating and medical labs the ability to use it to deliver analyses reports. The platform represents only a small use case of blockchain in healthcare, but it will allow the study of the feasibility of a larger system and its ability to solve the problems mentioned above.

Keywords: blockchain, healthcare, trust, consolidation, information access control
Resumo

Nos últimos anos as criptomoedas, tornaram-se tema de muito debate e interesse. Esta atenção foi causada pelo aumento de valor das criptomoedas. Muitas indústrias ficaram interessadas nas aplicações possíveis da tecnologia subjacente, blockchain. Blockchain refere-se a um livro-razão que mantém um registo permanente e imutável de transações que é replicado em todos os nós. A tecnologia blockchain oferece garantias de integridade dos dados, confiança na sua validade, transparência, descentralização e remove a necessidade de terceiras partes.

O setor da saúde é uma das maiores e mais valiosas indústrias do mundo. Abrange governos, empresas públicas, privadas e cidadãos. No entanto, existem alguns problemas na maneira como a indústria opera atualmente. A grande descentralização, a falta de comunicação entre profissionais e a falta de confiança na informação são apenas alguns destes problemas.

Considerando estes factos, o objetivo desta dissertação é criar uma plataforma para estudar a aplicabilidade da blockchain na solução dos problemas do setor de saúde. Essa plataforma armazenará dados médicos de pacientes, permitirá que eles controlem o acesso a esses dados, dará aos médicos a capacidade de aceder, adicionar e atualizar os dados da pessoa que estão a tratar e aos laboratórios médicos a possibilidade de entregar relatórios de análises. A plataforma representa apenas um pequeno caso de uso de blockchain na saúde, mas permitirá o estudo da viabilidade de um sistema maior e sua capacidade de resolver os problemas mencionados acima.

O trabalho foi desenvolvido na Accenture, uma consultora multinacional com grande foco na área da tecnologia. Outros dois colegas da Faculdade de Ciências também estavam a desenvolver projetos de blockchain com o mesmo objetivo, mas para indústrias diferentes. O meu supervisor na Accenture foi o Tiago Minchin, contando também com o Álvaro Silva na equipa da Accenture envolvida no projeto. Inicialmente, na fase de identificação de casos de uso, trabalhei em conjunto com os meus colegas da faculdade, mas quando ficou definido qual o caso de uso a desenvolver o trabalho tornou-se individual.

Para além da identificação do caso de uso, estudei várias possíveis implementações de blockchain de modo a escolher a mais correta para construir a plataforma. No final decidi utilizar o Hyperledger Fabric desenvolvido pela Fundação Linux. As razões para essa escolha foram o suporte de smart contracts, código que é usado para definir lógica de negócio na blockchain, e facto de o Fabric ser uma blockchain privada e poder ser permissionada, o que permite gerir as questões da privacidade dos dados. Os smart contracts podiam ser escritos nalgumas linguagens das quais eu escolhi o Node.js, pois era a melhor documentada e utilizada para o Hyperledger Fabric. Decidi também desenvolver a interface de utilizador utilizando Angular.

Após identificado o caso de uso, a blockchain e a linguagem de programação, procedi para o desenvolvimento das user stories, levantamento de requisitos, partes interessadas e arquitetura.

Com tudo isto, iniciei o desenvolvimento da plataforma, seguindo os exemplos e tutoriais fornecidos pelo Hyperledger. Em primeiro lugar criava os smart contracts e depois a interface de utilizador a funcionalidade. Em seguida fazia testes para tentar detetar erros, estes eram sempre funcionais. Quando estivesse tudo a trabalhar corretamente, repetia o processo para a funcionalidade seguinte. Quando terminei a implementação procedi à documentação de todo o código e à remoção de métodos, classes e módulos criados durante o desenvolvimento que já não eram necessários para a versão final.

Existem três tipos de utilizadores na plataforma: pacientes, médicos e laboratórios médicos. Os pacientes podem consultar o seu histórico médico, consultas, relatórios de análises e controlar o acesso a estes. Os únicos dados que estão sempre visíveis são as informações essenciais para poderem ser usadas em emergências. Os laboratórios usam a plataforma para adicionar relatórios de análises e podem consultar os relatórios que criaram. Os médicos, mediante autorização por parte do paciente, podem
iniciar consultas, nas quais vão poder registar a consulta, adicionar ou atualizar dados médicos e consultar o histórico médico, de consultas e relatórios de análises.

A interface de utilizador é uma aplicação que corre num navegador. A comunicação com a blockchain é feita através de um servidor, também ele escrito em Node.js. A blockchain como referido anteriormente, é a Hyperledger Fabric versão 1.4.0. É sobre esta implementação que instanciei os smart contracts, ou chaincode como são chamados no Fabric. Como referido anteriormente estes foram escritos em Node.js. Existem cinco chaincodes, medical information, analyses reports, medical history, consultation history e users. Medical Information é onde está definida a informação médica essencial e a lógica de negócio associada a esta. Analyses Reports define os relatórios de análises, a sua lógica e implementa as regras de acesso e criação. Medical History define os três tipos de dados guardados: alergias, medicamentos e condições médicas, a lógica de negócio e implementa o acesso a estes. Na consultation history está definida a consulta, as funcionalidades associadas a esta e as regras sobre o acesso e criação da mesma. Users define os três tipos de utilizador, lógica de negócio e acesso. Os utilizadores do tipo paciente mantêm uma lista dos médicos e instituições médicas autorizadas com os respetivos níveis de acesso aos vários tipos de dados médicos. Esta lista é usada pelos outros chaincodes para o controlo de acesso aos dados. Toda a plataforma, aplicação de navegador, servidor e blockchain com os chaincodes, correm localmente em contentores Docker instalados numa máquina virtual Ubuntu.

Este projeto foi a primeira vez que trabalhei com blockchain em geral e Hyperledger Fabric em particular, pelo que existem vários aspetos da minha implementação que faria de outra forma, agora que tenho mais experiência. O desenvolvimento do chaincode foi feito com base nos tutoriais disponibilizados para o Fabric, contudo pelos motivos mencionados anteriormente a organização do código não é a mais eficiente ou necessariamente a mais correta. Algo semelhante aconteceu na criação da interface de utilizador. Criação de interfaces gráficas de utilizador é algo que não foi o principal foco da minha instrução universitária pelo que esta foi também a primeira vez que utilizei Angular, o que levou a problemas de estruturação de código semelhantes ao que aconteceu com o chaincode.

Durante todo o trabalho recebi input da equipa da Accenture sobre as funcionalidades que deveriam ser implementadas e sugestões sobre a interface do utilizador e como estas deveriam ser estruturadas para serem mais intuitivas. Outra área onde houve bastante ajuda por parte da equipa da Accenture foi na identificação do caso de uso, providenciando conhecimento das áreas de negócios analisadas que ajudaram na seleção.

Com a plataforma terminada e usando todo o conhecimento que acumulei durante o estágio na Accenture, consegui concluir que existe utilidade para um sistema semelhante a ser implementado em maior escala e com mais funcionalidades. Possíveis casos onde isto seria vantajoso de implementar seriam num sistema público de saúde, i.e., em todos hospitais públicos de um país ou num operador de saúde privado com múltiplos hospitais ou clínicas. Para ambos os casos, a implementação iria substituir os sistemas atuais e daria, entre outros benefícios, aos pacientes controlo sobre quais os médicos dentro dessa rede que teriam acesso aos seus dados. Numa maior escala este sistema poderia ligar sistemas públicos e privados, com cada operador, estado e diferentes empresas gestoras de hospitais teriam sub-redes que estavam ligadas entre si permitindo o acesso aos dados médicos dos pacientes, mediante autorização, a todos os envolvidos. Esta solução ideal iria resolver os problemas mencionados anteriormente.

Palavras-chave: blockchain, saúde, confiabilidade, consolidação, controlo de acesso à informação
Table of Contents

Table of Tables ........................................................................................................................................... xvi
Table of Figures ........................................................................................................................................... xviii

Chapter 1. Introduction ................................................................................................................................. 1
  1.1. Motivation ........................................................................................................................................... 1
  1.2. Host Organization ................................................................................................................................. 1
  1.3. Accenture Team ................................................................................................................................... 2
  1.4. Summary ............................................................................................................................................ 2
  1.5. Contributions ..................................................................................................................................... 3
  1.6. Structure of the Document .................................................................................................................. 3

Chapter 2. Overview ..................................................................................................................................... 5
  2.1. Objectives ........................................................................................................................................... 5
  2.2. Context ................................................................................................................................................ 5
  2.3. Methodology ..................................................................................................................................... 6
  2.4. Planned Work ................................................................................................................................... 7
  2.5. Planning............................................................................................................................................. 7
  2.6. Execution .......................................................................................................................................... 8
  2.7. Planning vs Execution ......................................................................................................................... 10

Chapter 3. State of the Art ............................................................................................................................ 13
  3.1. Important Concepts ............................................................................................................................. 13
    3.1.1. Blockchain .................................................................................................................................. 13
    3.1.2. Nodes ......................................................................................................................................... 13
    3.1.3. Transactions ............................................................................................................................... 13
    3.1.4. Blocks ....................................................................................................................................... 13
    3.1.5. Ledger ....................................................................................................................................... 14
    3.1.6. Forks .......................................................................................................................................... 14
    3.1.7. Consensus Mechanisms ............................................................................................................. 14
    3.1.8. Smart Contracts ......................................................................................................................... 15
    3.1.9. Permissionless Blockchains ....................................................................................................... 15
    3.1.10. Permissioned Blockchains ....................................................................................................... 15
    3.1.11. Public Blockchains ................................................................................................................... 16
    3.1.12. Private Blockchains ................................................................................................................. 16
    3.1.13. Advantages ............................................................................................................................... 16
  3.2. Blockchain Implementations ............................................................................................................... 17
    3.2.1. Ethereum .................................................................................................................................... 17
    3.2.2. Hyperledger Fabric ..................................................................................................................... 17
    3.2.3. R3 Corda ..................................................................................................................................... 17
  3.3. Methodologies, Programming Languages and Tools ........................................................................ 17
    3.3.1. Agile .......................................................................................................................................... 17
    3.3.2. FURPS+ ..................................................................................................................................... 18
    3.3.3. JavaScript ................................................................................................................................. 18
    3.3.4. Node.js ...................................................................................................................................... 18
    3.3.5. Angular ...................................................................................................................................... 18
    3.3.6. TypeScript ................................................................................................................................. 18
4.10.1.2. Front-end................................ ................................ ................................ .......41
4.10.1.3. Server........................................................................................................42
4.10.2.  Analyses Reports ................................ ................................ ............................. 42
4.10.2.1. Chaincode ...........................................................................................43
4.10.2.2. Front-end...........................................................................................43
4.10.2.3. Server........................................................................................................45
4.10.3.  Medical History ......................................................................................... 45
4.10.3.1. Chaincode ...........................................................................................45
4.10.3.2. Front-end...........................................................................................47
4.10.3.3. Server........................................................................................................47
4.10.4.  Consultation History.................................................................................. 47
4.10.4.1. Chaincode ...........................................................................................48
4.10.4.2. Front-end...........................................................................................49
4.10.4.3. Server........................................................................................................50
4.10.5.  Users...........................................................................................................50
4.10.5.1. Chaincode ...........................................................................................51
4.10.5.2. Front-end...........................................................................................52
4.10.5.3. Server........................................................................................................54
4.10.6.  Information Access Control ...................................................................... 54
4.10.6.1. Medical Information Contract Class ....................................................... 55
4.10.6.2. Analyses Report Contract Class ............................................................... 56
4.10.6.3. Medical History Contract Class ............................................................... 56
4.10.6.4. Consultation History Contract Class ....................................................... 57
4.10.6.5. Users Contract Class ............................................................................. 58
4.10.7.  Shared Front-end Components................................................................... 58
4.11. Documents...................................................................................................... 60
4.12. Tests................................................................................................................ 60
4.13. Platform Evaluation........................................................................................ 61
Chapter 5.  Conclusion .......................................................................................... 65
5.1. Work Summary.............................................................................................. 65
5.2. Discussion....................................................................................................... 65
5.3. Future Work.................................................................................................... 66
Bibliography.......................................................................................................... 69
Table of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 2.1</td>
<td>Planning</td>
<td>7</td>
</tr>
<tr>
<td>Table 2.2</td>
<td>Timeline</td>
<td>9</td>
</tr>
<tr>
<td>Table 4.1</td>
<td>User Story 1 - Patient registering</td>
<td>26</td>
</tr>
<tr>
<td>Table 4.2</td>
<td>User Story 2 - Doctor registering</td>
<td>26</td>
</tr>
<tr>
<td>Table 4.3</td>
<td>User Story 3 - Medical Lab registering</td>
<td>26</td>
</tr>
<tr>
<td>Table 4.4</td>
<td>User Story 4 - Information control</td>
<td>26</td>
</tr>
<tr>
<td>Table 4.5</td>
<td>User Story 5 - Essential patient information access: doctor</td>
<td>27</td>
</tr>
<tr>
<td>Table 4.6</td>
<td>User Story 6 - Essential patient information access: patient</td>
<td>27</td>
</tr>
<tr>
<td>Table 4.7</td>
<td>User Story 7 - Essential patient information changes</td>
<td>27</td>
</tr>
<tr>
<td>Table 4.8</td>
<td>User Story 8 - Medical history access: doctor</td>
<td>27</td>
</tr>
<tr>
<td>Table 4.9</td>
<td>User Story 9 - Medical history access: patient</td>
<td>28</td>
</tr>
<tr>
<td>Table 4.10</td>
<td>User Story 10 - Medical history changes</td>
<td>28</td>
</tr>
<tr>
<td>Table 4.11</td>
<td>User Story 11 - Consultation history access: doctor</td>
<td>28</td>
</tr>
<tr>
<td>Table 4.12</td>
<td>User Story 12 - Consultation history access: patient</td>
<td>28</td>
</tr>
<tr>
<td>Table 4.13</td>
<td>User Story 13 - Consultation history changes</td>
<td>29</td>
</tr>
<tr>
<td>Table 4.14</td>
<td>User Story 14 - Adding analyses reports</td>
<td>29</td>
</tr>
<tr>
<td>Table 4.15</td>
<td>User Story 15 – Analyses report access: doctor</td>
<td>29</td>
</tr>
<tr>
<td>Table 4.16</td>
<td>User Story 16 – Analyses report access: patient</td>
<td>29</td>
</tr>
<tr>
<td>Table 4.17</td>
<td>User Story 17 – Analyses report access: medical lab</td>
<td>30</td>
</tr>
<tr>
<td>Table 4.18</td>
<td>Functionality Requirements</td>
<td>30</td>
</tr>
<tr>
<td>Table 4.19</td>
<td>Usability Requirements</td>
<td>31</td>
</tr>
<tr>
<td>Table 4.20</td>
<td>Reliability Requirements</td>
<td>33</td>
</tr>
<tr>
<td>Table 4.21</td>
<td>Performance Requirements</td>
<td>33</td>
</tr>
<tr>
<td>Table 4.22</td>
<td>Supportability Requirements</td>
<td>33</td>
</tr>
<tr>
<td>Table 4.23</td>
<td>Design Constraints</td>
<td>34</td>
</tr>
<tr>
<td>Table 4.24</td>
<td>Implementation Constraints</td>
<td>34</td>
</tr>
</tbody>
</table>
# Table of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Planning</td>
<td>8</td>
</tr>
<tr>
<td>2.2</td>
<td>Timeline</td>
<td>9</td>
</tr>
<tr>
<td>3.1</td>
<td>Example of a block [3]</td>
<td>14</td>
</tr>
<tr>
<td>4.1</td>
<td>Use Case - original</td>
<td>25</td>
</tr>
<tr>
<td>4.2</td>
<td>Use case – implementation</td>
<td>25</td>
</tr>
<tr>
<td>4.3</td>
<td>Platform Architecture</td>
<td>37</td>
</tr>
<tr>
<td>4.4</td>
<td>Peer</td>
<td>39</td>
</tr>
<tr>
<td>4.5</td>
<td>Blockchain Network</td>
<td>39</td>
</tr>
<tr>
<td>4.6</td>
<td>Medical Information Details Component</td>
<td>42</td>
</tr>
<tr>
<td>4.7</td>
<td>Analyses Report Create Component</td>
<td>44</td>
</tr>
<tr>
<td>4.8</td>
<td>Analyses Report Details Page Component</td>
<td>44</td>
</tr>
<tr>
<td>4.9</td>
<td>Analyses Report Search Component</td>
<td>45</td>
</tr>
<tr>
<td>4.10</td>
<td>Analyses Report Details Dialog Component</td>
<td>45</td>
</tr>
<tr>
<td>4.11</td>
<td>Medical History Details Component</td>
<td>47</td>
</tr>
<tr>
<td>4.12</td>
<td>Medical History Search Component</td>
<td>47</td>
</tr>
<tr>
<td>4.13</td>
<td>Consultation History Default Component</td>
<td>49</td>
</tr>
<tr>
<td>4.14</td>
<td>Consultation History Details and Create Component</td>
<td>49</td>
</tr>
<tr>
<td>4.15</td>
<td>Consultation History Details and Search Component</td>
<td>50</td>
</tr>
<tr>
<td>4.16</td>
<td>Consultation History Dialog Component</td>
<td>50</td>
</tr>
<tr>
<td>4.17</td>
<td>User Register Component - Step 1</td>
<td>53</td>
</tr>
<tr>
<td>4.18</td>
<td>User Register Component - Step 2</td>
<td>53</td>
</tr>
<tr>
<td>4.19</td>
<td>User Register Component - Step 3</td>
<td>53</td>
</tr>
<tr>
<td>4.20</td>
<td>User Login Component</td>
<td>53</td>
</tr>
<tr>
<td>4.21</td>
<td>User Profile Component</td>
<td>53</td>
</tr>
<tr>
<td>4.22</td>
<td>Patient Permission Component</td>
<td>53</td>
</tr>
<tr>
<td>4.23</td>
<td>Patient Add Doctor Permissions Dialog</td>
<td>54</td>
</tr>
<tr>
<td>4.24</td>
<td>Patient Update Organization Permissions Dialog</td>
<td>54</td>
</tr>
<tr>
<td>4.25</td>
<td>Allergy Create Dialog</td>
<td>59</td>
</tr>
<tr>
<td>4.26</td>
<td>Allergy Details Dialog</td>
<td>59</td>
</tr>
<tr>
<td>4.27</td>
<td>Medical Condition Create Dialog</td>
<td>59</td>
</tr>
<tr>
<td>4.28</td>
<td>Medical Condition Details Dialog</td>
<td>59</td>
</tr>
<tr>
<td>4.29</td>
<td>Medication Create Dialog</td>
<td>60</td>
</tr>
<tr>
<td>4.30</td>
<td>Medication Details Dialog</td>
<td>60</td>
</tr>
</tbody>
</table>
Chapter 1. Introduction

This chapter contains the motivation for the developed work, a short summary of it, contributions to both the healthcare industry and blockchain development, a contextualization of the work environment and team, as well as the structure of the rest of the dissertation report.

1.1. Motivation

With the advance of technology more and more industries are changing their business models and updating their digital infrastructure, ending their reliance on paper documents and moving to digital documents. An industry that seems to be lagging is healthcare. Given the importance healthcare has in society and the money the industry moves around the world every year, the reliance on paper and lack of communication between different institutions and physicians makes some of the procedures look archaic. This is in part due to governing regulations and laws that failed to keep up with new tools and technologies. However, there is considerable resistance to change by all involved members. Thanks to this it falls on the patients the responsibility to carry all their medical records and remember relevant information, like allergies or medication being taken. Given that, at times, interactions with medical professionals are made during duress, for example in a visit to an emergency room or after an accident, some of that information might not be properly transmitted which can cost someone their life.

Another problem people in healthcare have to deal with is individuals who falsify medical tests in order to receive prescription medicine. This causes doctors and other professionals to view results of tests they did not order with distrust.

Taking into consideration the issues presented above some in the healthcare industry have started to look into ways to modernize the industry. Some projects have been undertaken, trying to consolidate medical data. Though almost all of these projects rely on third party servers and systems to manages the data, which adds costs and creates privacy concerns about unwanted access to the information on the database.

The issue of data privacy is of the utmost importance for people. With scandals like Cambridge Analytica and others, many would be very skeptical of having their data added to yet another database where they cannot control who sees their private information.

Blockchain by its nature offers integrity guarantees, ensuring that no one can change data after the fact and authenticating the origin of that data, i.e. who added it. Since it is a distributed ledger and the copies of it are stored locally by every member of the network, this removes both the problems of cost to host the database and the lack of trust in this third party. Many blockchains also allow the implementation of smart contracts, pieces of code that can be used to manage access to the data, along with other business logic.

Many consulting firms, technology companies and industry insiders began considering blockchain as a possible solution to modernize healthcare for all the reasons listed above. Among those companies is Accenture.

1.2. Host Organization

The work was conducted at Accenture. Accenture is multinational consulting firm offering services in strategy, consulting, technology, digital and operations. It operates in 120 countries working across 40 industries. [1]
Chapter 1. Introduction

1.3. Accenture Team

At Accenture I was working under the direction of Tiago Minchin, as well as, Álvaro Silva, both of whom from now on will be referred to as “Accenture team”. There was no co-supervisor from Accenture in the dissertation due to changes in the governing legislation of who can be a supervisor and co-supervisor. Along with me, two other colleagues from the Faculty of Sciences of the University of Lisbon were also developing blockchain solutions for other industries. Some of the first tasks undertaken were made in tandem with one another but as soon as each identified the use case our work became separate.

1.4. Summary

The work for the dissertation encompassed all phases of software development. Initially the use case needed to be decided, to that effect multiple industries and their problems were analyzed. This potential use cases were placed on a matrix and evaluated. In the end, healthcare was chosen. Still the scope of the problem was far too big and relied on multiple stakeholders which made it impossible to create a solution during the nine months of the dissertation. As such in conjunction with the Accenture team the scope was narrowed to one that could be developed in the allotted time and in the work context described above.

With the use case selected and scope well defined, the next phase was the selection of the blockchain implementation to be used for the platform. A study of blockchain technology was also conducted concurrently with this and the previous stage of work. There were several requirements the for selection, it had to be a private and permissioned blockchain, there had to be support for smart contracts, the consensus mechanism had to be fault tolerant and it had to be open source. Other non-solution requirements, related more with my personal inexperience with blockchain, were the supported smart contract implementation languages and the existence of an active developer community where difficulties and questions could be answered. With these requirements and a list of potential blockchains with their specifications, Hyperledger Fabric was selected.

Following the blockchain selection, the next step was the identification of the stakeholders, requirements, user stories and the development of the architecture. Some of the requirements and architecture are not as detailed as they should due to the lack of experience with the technology and lack of good examples and resources to use as basis, due to the how recent blockchain development is.

Next the work moved to the implementation of the solution. To this effect, following the tutorials available on the Hyperledger Fabric website, the blockchain was setup. Then it was necessary to configure the network. With this last step, the network was ready to receive the chaincode (smart contract for Fabric). As such chaincode methods implementing each function were developed and instantiated on the blockchain. Simultaneously, the user interface was also being developed. After a new function was instantiated in the blockchain, the front-end that would allow users to interact with it was created. It was through this interface that the chaincode was tested. In order for the web-app to communicate with the blockchain a server was also created. When no more errors or bugs were detected, either on the front or back-ends, the next function would be developed. All the produced code was then documented.

The final phase was the evaluation of the platform. This was done to try and understand whether or not there was a use for blockchain solutions in healthcare to solve the problems mention above. This was done theoretically, i.e., there was no actual implementation of the platform in any real-world environment. The platform was analyzed on whether or not it could perform all the required functions and solve the problems in its scope. With the information from this analysis a conclusion on if a larger system based on the one created could solve the industry problems was reached.
1.5. Contributions

The work conducted in this dissertation has potential contributions for both the healthcare industry and blockchain development.

The most direct contributions to the industry that can be taken from the dissertation are its conclusions. These can be used in conjunction with other factors relevant to the specific case to decide on whether or not a blockchain solution is merited. The platform developed is a proof-of-concept and as such would be considered as a jumping off point for any real-world application of the technology in the industry. Despite the fairly small scope and scale of the platform it is still a practical implementation of a new solution implemented using innovative technology in an industry that is looking to modernize.

Much like the contributions for the industry, the field of blockchain development can also benefit from the conclusions of the dissertation. A fact that was observed during the use case identification stage is that, notwithstanding the interest in blockchain and the possible benefits of the technology for a myriad of industries almost all discussion is theoretical. This can be attributed to the fact that, the process of identifying a use case for implementation is not well defined and that leads to ideas where the cost of the technology would outweigh the benefits. Another problem is that the cases where it is advantageous to use blockchain are so large and complex that creating a proof-of-concept is too expensive and time consuming to interest developers, large and small. This was also a problem that appeared during the initial stages of work, as alluded to previously. To solve this, the scope of the problem was reduced to an implementable solution. It is this that is necessary for the development of blockchain proof-of-concepts, smaller solutions to test the technology in the area in question. The reason that this is not a standard practice is because sometimes in the small proof-of-concept cases blockchain loses its usefulness. It has to be taken into account that it is for the larger problem that blockchain use has to make sense. The implemented solution is meant to show what the technology can do in the context of the problem and serve has a foundation for a future system. Besides explaining how the use cases considered were evaluated and after selection how the scope was narrowed, the final product, the platform is an actual proof-of-concept that was implemented. While that might not seem like a valuable contribution, because of the mostly theoretical nature of potential solutions and ones that are not were created to test the technology without any specific use case in mind, a platform that can be as basis for future work is valuable for the field.

1.6. Structure of the Document

The document structure is as follows:

- Chapter 2 – describes the objectives of the work, its context, the methodology used, as well as, the planning, the final timeline and a discussion of the differences between the two;
- Chapter 3 – it includes the tools used, important blockchain concepts, related work and comparison between it and the developed solution;
- Chapter 4 – contains a detailed description/explanation of all the work from the use case selection to the evaluation of the final platform;
- Chapter 5 – summary of the work, it is conclusions, a critical commentary and discussion of future work that could be done to improve the platform.
Chapter 2. Overview

In this chapter the objectives and context of the dissertation are presented. The methodology used is explained. Finally, the initial planning, the actual timeline of development and a comparison of the two are also included.

2.1. Objectives

The objective of the dissertation is to determine whether or not it is advantageous to use blockchain to solve the problems identified in the healthcare industry. To this effect a blockchain powered platform was developed. It is used to store the medical history, essential information, consultation history and analyses’ reports of patients. The patients have an interface through which they can manage who sees and can add/update their information. Doctors use the platform to access authorized information about their patients to perform better diagnosis and treatments, as well as to add to/update the existing information for better accuracy in future treatments. Medical laboratories deliver the reports of analyses via the platform to ensure that they are made immediately available to both doctor and patients. After being built, the platform was be evaluated to determine the usefulness of the underlying technology for solving the problems discussed earlier.

Another objective is the creation of a proof-of-concept that, if determined to be useful, can be used as a basis for future implementations of a blockchain solution for medical data storage and handling.

2.2. Context

Interest in blockchain has been increasing over the past years. This has created a wide array of ideas of how the technology can change almost everything. From voting to supply chain management, airplane maintenance and many others. Many of these supposed solutions are almost exclusively concepts without any sort of practical development. In some of them blockchain is not even required, being more costly or complex than more established technologies.

This lack of real-world implementation also generates uncertainty surrounding blockchain. While there seems to be plenty of evidence that it can in fact be transformative for some cases, appearances can be deceiving. It is entirely possible, though unlikely, that blockchain is just an interesting idea with no practical and commercially viable application.

The technology has also gained a somewhat bad reputation for being the basis for cryptocurrencies. This created an association to the point where some think blockchain and bitcoin are interchangeable. The association is one that those promoting the technology are trying combat. This is because cryptocurrencies, and bitcoin in particular, are mostly known for their use in the Dark Web and illegal activities.

For many years it was this latter point that turned serious investors away from blockchain. However, when bitcoin and other cryptocurrencies had a surge in valuation many companies, governments and ordinary people started paying attention. Attention which was soon turned to the possibilities of the technology. Accenture is one those companies that decided to invest in blockchain.

As alluded to before, evaluating if blockchain is the best option is not a straight forward issue. There are no standardized decision trees or equivalent to be used to make the choice. While some have tried to create them, a part of it is still guess work and assumptions. The uncertainty of the decision
process coupled with the other issues discussed creates a problem for those that are trying to invest in
the technology.

Potential clients are also reluctant to invest given that the cost would go further than the price tag
of the system. The workforce would have to be trained to use this new paradigm and technology. The
lack of experienced professionals in the area also means that hiring people with knowledge of the tech-
nology is difficult. All of which is compounded with the aforementioned uncertainty around the usefulness
of blockchain. Meaning that those potential clients might decide against blockchain because they
fear ending up with an expensive system based on a technology that was abandoned.

Since this is a relatively new technology, few courses offer classes on blockchain software devel-
opment. This leads to a lack of credited professionals. Even the few courses or classes about blockchain
that exist fail to give the necessary preparation. This creates a disparity in the quality of the work devel-
oped. Considering the multiple different implementations of blockchain and the speed these are updated
and changed, being proficient in a version of one does not guarantee knowledge of another or even other
versions of the same implementation. The lack of certainty on the future of the technology is another
factor for the scarcity of experienced developers. Since some might be reticent of becoming experienced
in technology that might fade away.

It was considering all of these issues that the project was proposed by Accenture. With it, it is
expected to create an understanding of the technology and create a proof-of-concept that can be used to
evaluate the applications blockchain can have on an industry. In this case healthcare.

2.3. Methodology

Since I had no prior experience with blockchain, in particular Hyperledger Fabric, the tutorials
available at the Fabric website were an invaluable resource. All the smart contracts and the server were
created following the instructions on the site and based on the code of those tutorials. The first step in
creating a new smart contract was the definition of the data type, for example, in the Analyses Report
chaincode the first part was developing the AnalysesReport class that defines the attributes of that data
type and methods to interacted with it, such as those to get a specific attribute. Next the development
would move to the list class. This class stores a list of all objects of a type in the world state and is used
to interact with the StateList via a set of methods with all possible requests. The StateList is the class
that interacts directly with the ledger having methods for adding, updating and retrieving the data. The
final class to be created was the contract class. Here all the functionalities that are to be available for a
data type are defined in methods with the underlying business logic and access control.

Once the smart contract was created the server needed to be updated to include support for it. To
that effect new methods were created that received user requests, forwarded them to the appropriate
method on the contract.

With the server supporting requests to the new contract, user interfaces need to be created and
updated to allow the user to interact with the new data type and functionalities added to the blockchain.
Much like with blockchain, this was my first experience with Angular, so I followed online tutorials
extensively. First the new services, modules and components were generated. Then a data interface rep-
resenting the data type was created. Next in the development was the front-end interface. Followed by
adding to the components methods for responding to user interactions and sending the requested actions
to the service. On the service the methods that made the user requests to the server were added. Finally,
the routing between the different modules and components in the web app was updated to include the
newly created ones.

The final stage in the development was the testing. A series of functional test would be conducted
to identify errors and bugs on the code, both front and back-end. When a problem was detected it would
be debugged. After successfully resolving the issue, more tests would be conducted to make sure it was
truly solved, and no new ones appeared. Once nothing was detected, the development of that data type and functionalities would be completed and the process would start again for a new type.

2.4. Planned Work

The original plan to build the platform consisted of:

- User Story Creation – development of user stories to represent features and functionalities of the platform;
- Blockchain Evaluation – small comparative analysis of blockchains and selection of the best one for developing the system;
- Architecture Definition – creation of a well-defined architecture for the solution;
- Environment Construction – setting up the development and operation environment for the platform;
- Blockchain Development – setting up the blockchain that will serve as a basis for the entire system;
- Information Control Development – developing the information control logic;
- Patient Interface Development – creating the user interface and back-end for information control, sign-up and login by the patient;
- Doctor Interface Development – creating the interface and back-end for doctor sign-up, login, data requests, information updates and additions as well as back-end logic;
- Lab Interface Development – creating the interface and back-end for medical lab registration, login, adding reports and necessary logic;
- Document Production – production of supporting documentation;
- Tests, Platform Evaluation and Improvements – running functional tests on the platform, evaluating the platform and, time permitting, improving the system with new functionalities.

Given the relatively novelty of blockchain development and my personal inexperience on the subject it was decided with the Accenture team to leave some of the tasks less defined in the planning phase to allow for more freedom to change some aspects if deemed necessary during development.

Another decision reached with the help from the Accenture team was leaving open the possibility of implementing more functions besides the ones mentioned on the planned tasks above. These extra features would only be considered if everything planned has been successfully completed and the remaining time is deemed enough to even attempt. This is highly unlikely due to the reasons mentioned above.

2.5. Planning

Table 2.1 details the planning from the start date of the project October 12, 2018 to the end date July 19, 2019. Due to the reasons mentioned above some of these dates are subject to change. The largest task, the prototype development purposely as a three-day gap to give the development of the sprints some slack for easier changes to the plan.

<table>
<thead>
<tr>
<th>Task</th>
<th>Start Date</th>
<th>End Date</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case Definition</td>
<td>10/12/2018</td>
<td>11/20/2018</td>
<td>39</td>
</tr>
<tr>
<td>User Stories</td>
<td>11/21/2018</td>
<td>11/30/2018</td>
<td>9</td>
</tr>
</tbody>
</table>
### Chapter 2. Overview

#### Table 2.2

<table>
<thead>
<tr>
<th>Task</th>
<th>Start Date</th>
<th>End Date</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blockchain Evaluation</td>
<td>12/1/2018</td>
<td>12/10/2018</td>
<td>9</td>
</tr>
<tr>
<td>Architecture Definition</td>
<td>12/11/2018</td>
<td>12/21/2018</td>
<td>10</td>
</tr>
<tr>
<td>Platform Development</td>
<td>1/2/2019</td>
<td>4/12/2019</td>
<td>100</td>
</tr>
<tr>
<td>Sprint 1 - Environment Construction</td>
<td>1/2/2019</td>
<td>1/17/2019</td>
<td>15</td>
</tr>
<tr>
<td>Sprint 2 - Blockchain Development</td>
<td>1/18/2019</td>
<td>2/1/2019</td>
<td>14</td>
</tr>
<tr>
<td>Sprint 3 - Information Control Development</td>
<td>2/4/2019</td>
<td>2/19/2019</td>
<td>15</td>
</tr>
<tr>
<td>Sprint 4 - Patient Interface Development</td>
<td>2/20/2019</td>
<td>3/7/2019</td>
<td>15</td>
</tr>
<tr>
<td>Document Production</td>
<td>4/22/2019</td>
<td>5/2/2019</td>
<td>10</td>
</tr>
<tr>
<td>Tests, Platform Evaluation and Improvements</td>
<td>5/3/2019</td>
<td>7/19/2019</td>
<td>77</td>
</tr>
<tr>
<td>Report Writing</td>
<td>10/12/2018</td>
<td>7/19/2019</td>
<td>280</td>
</tr>
<tr>
<td>Preliminary Report</td>
<td>11/26/2018</td>
<td>12/12/2018</td>
<td>16</td>
</tr>
<tr>
<td>Final Report</td>
<td>4/22/2019</td>
<td>7/19/2019</td>
<td>88</td>
</tr>
</tbody>
</table>

*Figure 2.1: Planning*

#### 2.6. Execution

Table 2.2 presents the dates and duration of each task along with a chart of the timeline, figure 2.2.

While some the names of some tasks and sprints are different from the ones presented in the initial plan, they represent the same functions and actions as previously mentioned. The reason for the name, other changes and deviations from the original planning are explained in the next section of the report, Planning vs Execution.
### Table 2.2: Timeline

<table>
<thead>
<tr>
<th>Task</th>
<th>Start Date</th>
<th>End Date</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case Definition</td>
<td>10/12/2018</td>
<td>11/20/2018</td>
<td>39</td>
</tr>
<tr>
<td>User Stories</td>
<td>11/21/2018</td>
<td>11/30/2018</td>
<td>9</td>
</tr>
<tr>
<td>Blockchain Evaluation</td>
<td>12/1/2018</td>
<td>12/10/2018</td>
<td>9</td>
</tr>
<tr>
<td>Architecture Definition</td>
<td>12/11/2018</td>
<td>12/21/2018</td>
<td>10</td>
</tr>
<tr>
<td>Platform Development</td>
<td>1/2/2019</td>
<td>4/26/2019</td>
<td>114</td>
</tr>
<tr>
<td>Sprint 1 - Environment Construction</td>
<td>1/2/2019</td>
<td>1/17/2019</td>
<td>15</td>
</tr>
<tr>
<td>Sprint 2 - Blockchain Development</td>
<td>1/18/2019</td>
<td>2/11/2019</td>
<td>24</td>
</tr>
<tr>
<td>Sprint 3 - Medical Information Functions and Interface</td>
<td>02/12/2019</td>
<td>02/19/2019</td>
<td>7</td>
</tr>
<tr>
<td>Sprint 4 - Analyses Reports Functions and Interface</td>
<td>02/12/2019</td>
<td>02/25/2019</td>
<td>13</td>
</tr>
<tr>
<td>Sprint 5 - Medical History Functions and Interface</td>
<td>02/26/2019</td>
<td>03/08/2019</td>
<td>10</td>
</tr>
<tr>
<td>Sprint 6 - Consultation History Functions and Interface</td>
<td>03/01/2019</td>
<td>03/11/2019</td>
<td>10</td>
</tr>
<tr>
<td>Sprint 7 - CA Setup</td>
<td>03/12/2019</td>
<td>03/22/2019</td>
<td>10</td>
</tr>
<tr>
<td>Sprint 8 - User Functions and Interface</td>
<td>03/23/2019</td>
<td>04/02/2019</td>
<td>10</td>
</tr>
<tr>
<td>Sprint 9 - Information Access Control Logic and Interface</td>
<td>04/03/2019</td>
<td>04/13/2019</td>
<td>10</td>
</tr>
<tr>
<td>Sprint 10 - Interface Improvements</td>
<td>04/23/2019</td>
<td>04/26/2019</td>
<td>3</td>
</tr>
<tr>
<td>Document Production</td>
<td>04/29/2019</td>
<td>05/31/2019</td>
<td>32</td>
</tr>
<tr>
<td>Tests, Result Evaluation and Improvements</td>
<td>02/12/2019</td>
<td>05/31/2019</td>
<td>108</td>
</tr>
<tr>
<td>Report and Thesis</td>
<td>10/12/2018</td>
<td>07/19/2019</td>
<td>280</td>
</tr>
<tr>
<td>Preliminary Report</td>
<td>11/26/2018</td>
<td>12/12/2018</td>
<td>16</td>
</tr>
<tr>
<td>Thesis</td>
<td>05/08/2019</td>
<td>07/19/2019</td>
<td>72</td>
</tr>
</tbody>
</table>

### Figure 2.2: Timeline
2.7. Planning vs Execution

As it was alluded to in the previous section and is observable after comparing the planning and execution sections there were deviations and alterations from the original plan during implementation. This project was my first time using and studying blockchain, as such there was no personal history that could be used to create the planning. Since this is a new technology there were no projects with a planning to be studied at the host organization. Due to these factors, the plan was developed as that of any project for a similar non-blockchain platform, with slacks and enough room for change should it be required.

The first difference that can be observed is the increase in the duration of the second sprint of ten days. This is because to properly setup the blockchain, along with the correct Docker images, configuration of the network, more time was required. To do this, the tutorials offered on the Hyperledger Fabric website were followed. These tutorials covered not only setting up the network but also how to develop smart contracts. Besides these resources made available by the team behind Fabric, community pages and other online resources explaining how to setup and develop on Fabric were also consulted.

After doing the tutorials and learning more about Fabric and smart contract development it was clear that developing the access control as the first function would be impractical. This because it would complicate development and testing of all other features and also there would be no defined data or user types on the blockchain to take advantage of the features. As such, this sprint was moved to last. Moreover, the sprint was also split into three:

- **Sprint 7 - CA Setup**: setting up the certificate authority to generate user credentials upon sign-up and changing the server and related methods to use these new credentials;
- **Sprint 8 - User Functions and Interface**: developing the smart contracts for the user types and related functions, the user interfaces for sign-up, login, changing all the front-end to adapt to the current user, based on its type, and adding support for these new user related requests to the server;
- **Sprint 9 - Information Access Control Logic and Interface**: implementing the information access control. This was done not by a specific smart contract but by adding code to previously developed smart contracts. A front-end interface that allows patients to perform this control was also created.

With all of these new tasks the original fifteen-day sprint turned into three sprints totaling thirty days.

During the planning phase the idea would be to develop the code, both back and front-end with a focus on the user. This meant implementing all functions and developing the user interface for a user type and then moving on to the next. However, much like for the access control, this strategy was soon proven incorrect. Instead software was developed based on the type of data. Given this, planning sprints four, five and six were replaced with sprints three, four, five and six as they appear in the timeline in the previous section. In more detail:

- **Sprint 3 - Medical Information Functions and Interface**: development of the medical information smart contract, server support for requests and front-end interface for interactions with it;
- **Sprint 4 - Analyses Reports Functions and Interface**: development of the analyses reports smart contract, server support for requests and front-end interface for interactions with it;
- **Sprint 5 - Medical History Functions and Interface**: development of the medical history smart contract, server support for requests and front-end interface for interactions with it;
- **Sprint 6 - Consultation History Functions and Interface**: development of the consultation history smart contract, server support for requests and front-end interface for interactions with it.
Chapter 2. Overview

Giving that some of the code developed for each of these smart contracts and front-end is very similar to one another the sprints sometimes overlapped.

A final short sprint to fix some bugs, implement a few features, such as saving the hash of the user password in the blockchain instead of the actual password, and make small adjustments to the front-end was also added to the end of the platform development task.

The document production task’s goal remained unchanged, documenting the entire code and creating a presentation PowerPoint for Accenture with a short demonstrative video. This presentation was continuously improved with input from the Accenture team, which is the reason for this task to take up the entire month of May.

The testing task was also changed, to be concurrent to the development sprints of the smart contracts and front-end. After both were completed a series of functional tests were performed to spot bugs, errors and other problems. These would be fixed, and the tests repeated until no more issues were detected. After the end of the development task, a series tests with users were. It was decided not to implement any extra features because it would be too time consuming and the platform in its current state already permitted conclusions to be taken. This task was also ended in May to allow the remaining time to be dedicated to the production of this report. A decision that was made with the Accenture team.

Despite all the changes from the initial plan, all the features and functionalities of the platform were implemented, and the development task’s duration didn’t suffer an excessive increase.
Chapter 3. State of the Art

In this chapter there is a presentation of important concepts regarding blockchain that are needed to understand the technology, as well as, the tools, programming languages and methodologies used. All the considered blockchain implementation are also described here. Related systems and a comparison between them and the one developed for the dissertation is also included.

3.1. Important Concepts

3.1.1. Blockchain

Blockchain or distributed ledger technology describes a set of transaction records grouped into blocks. Each block is linked to the previous one using cryptography. The entire chain is stored in every node.

3.1.2. Nodes

A node on a blockchain refers to a client running an application that allows it to access and act upon the network. These actions can be executing transactions, mining, etc. Nodes are also responsible for managing the blockchain using peer-to-peer protocols and consensus mechanisms.

3.1.3. Transactions

A blockchain transaction is an action taken by a node in the blockchain. This action can be the transfer of cryptocurrency between peers, trade of the ownership of tokens or the update of the data stored in the blockchain.

Like the structure of the blocks, the type of transactions is only limited by what the developers stipulated when creating the blockchain. [2]

The verification of each individual transaction is also dependent on implementation of the blockchain.

3.1.4. Blocks

When a predefined number of transactions have been validated they are grouped in a block and added to the blockchain. Every block contains the hash value of the previous one in the chain, a timestamp and the data of every transaction contained in it. The transaction data is stored in Merkel Trees. All three of these elements are in the block header. It is the fact that every block contains the hash of the previous block that creates the continuous chain and allows for the trace back of every transaction stored in the blockchain going back to the first one.

The first block of a blockchain is the genesis block. Since there is no block prior its data is hard coded by the developer.
In the image above, it is possible to see all the elements referenced as well as others like the block nonce. Given the numerous implementations of blockchain the structure of the blocks tends to differ.

After being create a block is only added to the chain after a consensus is reached by the nodes. This mechanism can vary depending on the blockchain implementation, examples of consensus are proof-of-work and proof-of-stake.

3.1.5. Ledger

A copy of the blockchain is kept by each node. It is because the copies of the ledger are distributed across the network that blockchain technology is also called distributed ledger technology (DLT).

This can cause issues, because different nodes can have different versions of the ledger which leads to problems and inconsistencies when new blocks are being validated. Solving these problems is another of the reasons why consensus mechanisms are so important.

3.1.6. Forks

A blockchain fork happens when there is a divergence in the current status of the chain. This can be because nodes have different versions of the ledger. Usually this is resolved with consensus mechanisms. Forks come in two types, soft forks (when rules change but valid blocks under these rules are still valid under the old ones), and hard forks (when rules change but valid blocks under these rules are invalid under the old ones).

3.1.7. Consensus Mechanisms

To ensure that new blocks are not added by malicious nodes, there are consensus mechanisms to approve their addition to the chain. These mechanisms are also used to keep the most up to date version of the ledger in every node, by controlling the updates of those ledgers. Since blockchain is a decentralized system there is no owner or authority that can declare a certain version of the data the true and correct one. There are many advantages to this, namely that a malicious owner is not manipulating the data or resources of the network to their benefit. However, without this single source of truth, nodes need some way to agree on what is the correct version of the truth. This is where consensus mechanisms come in.

While the number and variety of consensus mechanisms is far too great to do an exhaustive analysis, the following are some of the most common:

- Proof-of-Work: it is based on the idea of trying to solve a very complex mathematical problem with a highly verifiable result. Nodes have to solve a problem associated with a
block to verify it. They do it by lending their computer’s CPU to making the calculation to solve the problem. This problem’s complexity is related to the difficulty of a block, an attribute set in the header of the block. The nodes that work to solve this problem are called miners and the process is called mining. The most famous example of proof-of-work application is Bitcoin, where miners are rewarded with a certain amount of the cryptocurrency if they contribute to the solution. The guiding principle is that the correct version of the ledger is the one with the most work done; [4]

- Proof-of-Stake: in this algorithm instead of being the node with the most work that gets to decide the most up to date version of the chain, it is chosen one of the nodes with the most stake in the decision. This stake is generally the amount of cryptocurrency held by a node. The more stake a node has, the more likely it is that it will be selected. There are some variations of this selection mechanism, such as randomized block selection, coin age-based selection, delegated proof-of-stake, randomized proof-of-stake; [4]

- Practical Byzantine Fault Tolerant Mechanism: blockchains like many other systems are vulnerable to byzantine failures. These happen when a certain number of nodes stop working properly or purposely act with malicious intent. As the name indicates, byzantine fault tolerant systems can function correctly despite the malfunctioning or malicious nodes, as long as they do not exceed a certain number. These mechanisms work by having nodes trade messages and comparing them. When a certain number of identical messages is reached, then the network achieved a consensus and the block is validated and added to the chain. [5]

All three of these mechanisms have some disadvantages, so blockchain developers have to be very careful in choosing them. Proof-of-Work’s hashing problems are extremely computationally and energy intensive creating a problem that could ultimately prevent blocks from being validated. In proof-of-stake, while it does not have the CPU and energy consumption of the previous system, in case of a fork in the blockchain nodes lose the incentive to keep stake in multiple chains, which would lead, in extreme cases to an impossibility in block validation. There is also a problem of double spending. [4] In the case of practical byzantine fault tolerant mechanisms, the message trading process can be slow which might lead to problems of slow block validation. [5]

3.1.8. Smart Contracts

Smart contracts are pieces of code that implement business logic on the blockchain. This can range from charging for the use of a resource, immediately sending updates to nodes when certain actions are performed on the chain, controlling access to information and data, to name a few. Not all blockchains support the implementation of smart contracts.

3.1.9. Permissionless Blockchains

A permissionless blockchain is one where anyone can sign up and fully participate without the need for approval by members. Bitcoin is the most obvious example of this type of implementation. In this blockchains all nodes can make transactions, create smart contracts (if supported), mine or invest stake in verifying blocks and be rewarded for those actions. [6]

3.1.10. Permissioned Blockchains

In contrast to permissionless blockchains, in permissioned blockchains entry into the network is restrict only to certain members. The actions nodes are allowed to perform also depend on what
permissions they are given. Nodes in these chains can only see certain transactions. It is this ability to control who joins the chain, what operations each member can perform and the transactions they can see that makes permissioned blockchains the preferred type for corporations, governments and businesses. [6]

3.1.11. Public Blockchains

Public blockchains refer to the ones where anyone can join without permission. While this might sound similar to permissionless blockchains, public is only about the ability to join the network. What a node can do and see will depend on the implementation. Though public blockchains tend to also be permissionless. [7]

3.1.12. Private Blockchains

A private blockchain is a network in which membership is controlled by some sort of authority, such as the creator of the chain. Like public blockchains, private networks have no obligation to be permissioned, though that is the norm. A private blockchain can allow every member to use all of its functions and access/see every transaction that occurs in it. Hyperledger Fabric is an example of a private blockchain. [7]

3.1.13. Advantages

DLTs have multiple advantages, namely:

- Immutability: once a block is added to the chain, changing the data of any of its transactions is virtually impossible. This happens because each block is cryptographically linked, so to change something, it’s hash value will also change and the next block’s header section with the hash of the previous block would no longer point to the changed block, making it easy to detect manipulation. To prevent this from happening it would be necessary to change the previous block hash of the next block for it to point to the changed block. However, this would also change that block’s hash, so the process would have to be repeated until the most recent block on the chain. While doable, the amount of computing power makes this a near impossibility.
- Transparency: regardless of the type of blockchain it is always possible for all or at least some of the members to access the records of every transaction. This allows for greater transparency because it is easy see what node made a certain transaction, making it easy to audit the chain.
- Trust: due to its guarantees of immutability and transparency blockchain gives nodes trust in the data and validity of the transactions. Not to mention that every node as the same source of truth.
- Data Access Control: using smart contracts and private permissioned blockchains it is possible to manage what nodes can see and do on the network.
- Decentralized: the decentralized nature of blockchain makes it far more resilient to attacks because unlike traditional databases there is no single point of failure.
- Eliminates Third Parties: since it creates trust in the authenticity and treatment of the data, while also allowing it to be processed in the network, blockchain removes the need for third parties to authenticate the data. Due to the fact that the ledger is stored in every node of the network, it also eliminates the need for third parties to store the data.
3.2. Blockchain Implementations

3.2.1. Ethereum

Much like Bitcoin, Ethereum is also a public blockchain and it as its own cryptocurrency, the ether. While Bitcoin was designed around bitcoin, its namesake currency, Ethereum can be used to build a wide variety of distributed applications with or without cryptocurrencies. It even allows developers to create their own currency for trading publicly or only in a specific application. [8] Unlike Bitcoin, smart contracts are a feature of Ethereum greatly increasing the capabilities of what can be done with this blockchain. Its creators meant Ethereum to be simple, fast to use, modular, universal and agile. The consensus mechanism utilized is proof-of-work, but the developers are working on implementing proof-of-stake to increase speed to which transactions and blocks are validated.

3.2.2. Hyperledger Fabric

Hyperledger Fabric is an implementation of Hyperledger, a DLT being developed by the Linux Foundation, whose objective is creating an open-source platform for businesses. Fabric is an IBM project and the main focus is the running of smart contracts, called chaincode. [9] It has a pluggable consensus mechanism, which gives developers greater freedom in choosing the system that better suits their solution. Currently it uses practical byzantine fault tolerant mechanisms to achieve consensus. Hyperledger Fabric is private giving it even greater privacy, by controlling who can join the network. With smart contracts it can also be permissioned. Hyperledger has no native currency and none can be created. However, using chaincode it is possible to develop tokens, but these are very different form bitcoin or ether.

3.2.3. R3 Corda

Corda was developed for the financial services industry. Just like Fabric it is a private DLT implementation that supports smart contracts and can also use them to manage access and operations by its members, allowing it to be permissioned. [10] In terms of consensus mechanism, these are generally at an individual and not global level and are defined during the implementation of the network and not predefined by the framework. A unique feature of Corda, at least with regards to the other blockchains in this list, is the fact that the only parts privy to a transaction are the ones involved in validating it. It was design with highly regulated industries in mind and to allow for institutions to communicate and work with each other without a central authority in the middle. Another feature of its design is the ability to add special nodes for regulators and observers. Smart contracts are used to directly mirror regulations, legislation, procedures and legal documents. In a similar fashion to Hyperledger, Corda does not include any cryptocurrency.

3.3. Methodologies, Programming Languages and Tools

3.3.1. Agile

The planning of the work was made using the Agile method for software development, because it offers more freedom and tolerance to changes and problems. It is these characteristics that make Agile ideal for creating a plan for software development using new technologies.
This methodology is based on the following ideas: greater importance in the people and interactions rather than the tools and processes; more focus on a working product than on extensive documentation; involving the customers during the development instead of relying only on what was negotiated in the contract; being responsive to changes and not being restricted by the plan. [11] The ideas mentioned above, and twelve principles of Agile software were published in 2001 by a group of seventeen signatories. Their document the Agile Manifesto became the basis for this methodology.

It should not be interpreted with it that procedures, tools, plans and others are not important, but that they ought not restrict development nor restrict flexibility to that address problems.

In Agile work is divided in sprints, short work periods that produce a something “tangible” in the end. [12] This division into sprints is due to the incremental nature of the methodology, i.e. with each task’s conclusion the value of the currently completed solution will increase. The sprints also allow for a fast identification of problems ensuring that they will not go unnoticed until very late in the project where resolving them would be more expensive.

3.3.2. FURPS+

To identify and classify the requirements I used FURPS+. This model is helpful because it includes not only the functional requirements, extracted from the user stories of the Agile method, but also the nonfunctional ones.

FURPS+ is a software quality attribute classifier model. It was developed by Hewlett-Packard. It stands for: functionality; usability; reliability; performance; supportability. The “+” refers to the following requirements: design constraints; implementation constraints; interface constraints; physical constraints. [13]

3.3.3. JavaScript

JavaScript is an object-oriented programming language primarily used in the development of web pages. It is a client-side language that allows for the scripting of behavior of web sites and other web-based applications. [14] JavaScript is not a typed language, i.e. it does not have the rigid data types of Java or C.

3.3.4. Node.js

The smart contracts and server were written in Node.js. This was done due to the better support of the Node SDK for smart contracts in Hyperledger Fabric. Node.js has as its core JavaScript. It is ideal for developing network applications due it’s asynchronous event driven runtime nature. Node.js was developed to facilitate scalability of applications. [15]

3.3.5. Angular

The entire front-end of the platform was written using Angular. This is development platform for applications. The languages used were TypeScript for the logic and HTML for the user interfaces. [16]

3.3.6. TypeScript

TypeScript is a typed implementation of JavaScript. Syntactically there is no difference between the two, except the ability to add variable and attribute types on TypeScript. It was developed by Microsoft. [17]
3.3.7. HTML

The front-end user interfaces were written in HTML. HTML stands for HyperText Markup Language. This language is ubiquitous in web development, being the basis for many user interfaces. [18]

3.3.8. CSS

CSS, Cascading Style Sheets, is used to define the appearance of an HTML page. With this language it is possible to stipulate what every element of a document will look like rendered, from the background to the size and color of the font. [19]

3.3.9. JSON

JavaScript Object Notation, JSON, is used by JavaScript to define complex objects in an easy to understand format. It uses a name:value structure to store one or more attributes.

3.3.10. YAML

The blockchain network configuration files were written in YAML, YAML Ain't Markup Language. This is a data-oriented language structure. It is similar to JSON, also using a key:value structure for the data. [20]

3.3.11. Docker

The network and its components, like the smart contracts and nodes, run in Docker containers. A container is a standard software unit that contains all the code and dependencies of an application. [21] Multiple elements of the network are docker images, executable packages with all that is necessary for running the code inside.

3.3.12. Visual Studio Code

All the code was written on the Visual Studio Code editor. Visual Studio Code or VS Code is a code editor developed by Microsoft. Similar to other editors like Eclipse, it offers multiple features to assist in the development process, but for a vast array of languages no just Java. It has the ability to install plugins, extensions and extra features. [22]

3.3.13. IBM Blockchain Platform for Visual Studio Code

One of the VS Code features that was used during development was the IBM Blockchain Platform Extension. It was used for creating, packaging, installing and instantiating the smart contracts on the respective nodes. [23]

3.3.14. CouchDB

The state database in the peers uses CouchDB. CouchDB stores the data in JSON files and allows it’s for easy access, manipulation and querying. [24] It possesses incremental replication mechanisms that permit data distribution.
3.4. Related Work

3.4.1. MedRec

MedRec is a system developed by Ariel Ekblaw, Asaph Azaria, John D. Halamka, MD and Andrew Lippman with MIT Media Lab and Beth Israel Deaconess Medical Center. It was designed to store and manage electronical health records (EHR) in a decentralized system. The goal is to give patients access to all their health records. [25] MedRec was design with modularity in mind so it can work with existing systems of healthcare providers. It uses proof-of-work and in order to “attract” miners, such as researchers and public health authorities. These rewarded for mining by being given access to medical statistics and data. Any identifiable information of these data is always removed from what is sent to the miners. It was implemented using Ethereum.

3.4.2. DokChain

A California company PokitDok developed DokChain for the financial and healthcare industries. Its goal is to take advantage of blockchain’s strengths to create a safe network for all involved in healthcare to communicate. Healthcare providers and others involved would be connected via DokChain replacing the current method. [26]

3.4.3. Patientory’s Healthcare Platform

This start-up is developing a system that gives patients the ability to control access to their electronical medical records. A hospital, clinic or other healthcare provider with access can create new information about the patient. To be able to use the platform a patient has to own PTOY, a cryptocurrency created by Patientory. The goal of their solution is to create a secure system where medical records can be securely accessed by the relevant parties and to make patients the middleman managing access to their records. [26]

3.4.4. GemOs

It was developed by Gem a California blockchain start-up with focus on healthcare and supply-chain management. They developed the Gem Healthcare Platform on Ethereum to give patients the ability to manage access to their medical data and create a secure place to keep all medical records and add new ones. GemOs was design for claims management. It will, according to Gem, expedite the process by giving patients and providers access to updated records. [26]

3.4.5. KSI in Healthcare

Guardtime is a tech company from Estonia known for its blockchain based security systems. They use keyless signature infrastructure (KSI) in their system to guarantee security. In 2016 the company partnered with Estonia’s government to provide security to the countries health records. [26]

3.5. Comparison to Discussed Solutions

When comparing the developed platform with the similar systems presented above some conclusions can be reached:
• MedRec: both solutions have some common characteristics, specifically, creating a consolidated healthcare system where patients can have all their data. However, MedRec does not seem to focus as much on giving other parties access to the data for diagnosis use. Moreover, that access is more of a reward for mining in order to validate the data;

• DokChain: this network and the platform have in common the base idea of creating a secure “place” for healthcare providers and others in the industry to communicate. But while DokChain focuses only on the communication part, the platform has communication, via access of a history and was designed with a permanent registry of patients’ health data in mind;

• Patientory’s Healthcare Platform: both platforms have the same core idea, giving patients the capability of managing who has access to their data. The developed system however does not require any cryptocurrency to use;

• GemOs: just like the previous system GemOs has several similarities with the platform developed for the dissertation. Despite this, GemOs uses the medical data for the purpose of claims management, something that is totally outside of the current scope of the system;

• KSI in Healthcare: making a good comparison between the implemented system and what they are implementing in Estonia is hard, because the source did not mention what features will be maintained or added in the new solution, referencing only the way Estonians currently access the existing healthcare system.
Chapter 4. The Work

This chapter presents a detailed description of the work from the use cases considered to the evaluation of the platform.

4.1. Use Case Selection

In this section the process for identification of potential use cases and evaluation criteria are presented. In this initial phase the work was performed with two other colleagues from the Faculty of Sciences. There was also heavy assistance from the Accenture Team, that provided knowledge of the industries in which the use cases were inserted.

4.1.1. Identification

The process of identifying use cases consisted of extensive reading of internal Accenture documents detailing potential uses of blockchain in various industries. These varied from developed systems to theoretical applications. To complement the use cases identified from the Accenture documents, multiple articles detailing blockchain applications were read and from these more cases were identified. In the end the list counted fifty-two use cases, some were variations of others.

These cases covered multiple industries and areas, such as financial services, healthcare, product sales, public services, resources and some were cross-industry. Some examples are:

- inventory management – cross-industry;
- ownership titles – cross-industry;
- peer-to-peer money transfers – financial services;
- insurance – financial services;
- stocks- financial services;
- consolidated healthcare – healthcare;
- medical research sharing platform – healthcare;
- e-tickets – products;
- anti-counterfeiting – products;
- airplane maintenance records – airlines;
- digital identity – public services;
- education history – public services;
- pensions – public services;
- e-voting – public services;
- e-census – public services.

4.1.2. Evaluation

With the list of possible use cases finished, the work moved to their evaluation. To perform this evaluation a set of criteria was selected. These were provided by the Accenture team. The criteria were:

- ease of implementation;
- relevance to the company’s strategy or industry;
- blockchain relevance for the use case;
- legal, regulatory and compliance aspects & complexity;
- number and types of actors involved;
interoperability with existing and third-party systems;
• openness/favorability of the internal and/or external business partners;
• market exposure;
• interest in the use case.

Before the criteria were used these were analyzed to ensure that they were adequate and would produced logical and useful results. This analysis of the criteria was done using the knowledge of block-chain gained to that point along with some business knowledge learned during the identification phase from the documents, articles and the Accenture team. The criteria were concluded to be adequate for the evaluation of the use cases. They evaluate the use case on every major aspect from the value of the technology for the case, ease of implementation and interoperability with existing systems, to the market exposure, legal compliance and the new system’s relevance for the strategy of the industry. This creates a good balance between the technology and the business sides.

The last criterion was added by suggestion of the Accenture team to represent how interested we, me and my colleagues, were in developing each use case. Each criterion had a weight associated with it to ensure that the ones more relevant for the project and its scope, had more impact on the ranking of the use case.

After the evaluation the five highest ranked use cases were:

1. e-tickets;
2. consolidated healthcare;
3. insurance;
4. pensions;
5. digital identity.

I opted to develop the consolidated healthcare use case because I found it to be the most interesting.

4.2. Selected Use Case

The selected use case consisted of a ledger shared by every public and private healthcare provider. These entities would access the information they needed, if provided access, to conduct their diagnosis and perform treatments, as well as, add any new data resulting from said diagnosis and treatments. The ledger would store, for each patient, essential medical information (height, weight, allergies, etc.), medical history (previous diseases, injuries, operations, treatments, professional responsible for the treatment, etc.), vaccination history, consultation history (record of every visit to a doctor), ER visits (similar to consultation history but about visits to the Emergency Room) and analyses reports. These data would be updated, or new data added every time there was an interaction with a healthcare provider or medical laboratories.

Patients would own all information related to them and have the ability to control access to it, choosing which organizations and professionals have access to it.

This consolidated system would look like the following:
However, the scope of this use case was too big for it to be a proof-of-concept to be developed in the context of a master’s dissertation by a single person with no prior experience using the technology. As such with the help of the Accenture team the scope was narrowed to a feasible case.

In the new use case the patient and medical labs were maintained with the same actions. The healthcare providers, though, were simplified to doctors with the same functions as before. The types of data stored in the blockchain were also changed, vaccination reports and ER visits are no longer features and the medical history contains only a record of previous allergies, medical conditions and medications.

This has the following structure:

**4.3. Stakeholders**

The platform’s end users are: patients, doctors, medical lab technicians. The clients will be medical laboratories, private and public healthcare providers that employ the doctors. In terms of developer and system architect, there was only me holding both roles. The position of project manager for the development of the platform was held by the members of the Accenture team.
## 4.4. User Stories

### Table 4.1: User Story 1 - Patient registering

<table>
<thead>
<tr>
<th>ID: H1</th>
<th>User Story: Patient registering</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>As a new patient, I want to register in the system, so that I can have my data stored in the blockchain and manage who has access to it.</td>
</tr>
<tr>
<td>Acceptance Criteria:</td>
<td>• The new patient is registered into the system with the correct information.</td>
</tr>
<tr>
<td>Assumptions:</td>
<td>• Patient is not registered.</td>
</tr>
<tr>
<td>Outcomes:</td>
<td>• New patient is added to the system; • All patient data is saved in the system.</td>
</tr>
</tbody>
</table>

### Table 4.2: User Story 2 - Doctor registering

<table>
<thead>
<tr>
<th>ID: H2</th>
<th>User Story: Doctor registering</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>As a doctor, I want to register in the system, so that I can access my patients’ medical data and add new one to better perform diagnosis and treatments.</td>
</tr>
<tr>
<td>Acceptance Criteria:</td>
<td>• The new doctor is registered into the system with the correct information.</td>
</tr>
<tr>
<td>Assumptions:</td>
<td>• Doctor is not registered.</td>
</tr>
<tr>
<td>Outcomes:</td>
<td>• New doctor is added to the system; • All doctor data is saved in the system.</td>
</tr>
</tbody>
</table>

### Table 4.3: User Story 3 - Medical Lab registering

<table>
<thead>
<tr>
<th>ID: H3</th>
<th>User Story: Medical Lab registering</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>As a medical lab manager, I want to register the lab in the system, so it can upload the reports of the patient’s analyses to the blockchain where they will be accessed to perform diagnosis and treatments.</td>
</tr>
<tr>
<td>Acceptance Criteria:</td>
<td>• The new lab is registered into the system with the correct information.</td>
</tr>
<tr>
<td>Assumptions:</td>
<td>• Lab is not registered.</td>
</tr>
<tr>
<td>Outcome:</td>
<td>• New lab is added to the system; • All lab data is saved in the system.</td>
</tr>
</tbody>
</table>

### Table 4.4: User Story 4 - Information control

<table>
<thead>
<tr>
<th>ID: H4</th>
<th>User Story: Information control</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>As a patient, I want to manage access to my data, so that I control who can see it and what they can access, update and add.</td>
</tr>
<tr>
<td>Acceptance Criteria:</td>
<td>• Information access permissions are updated according to patient’s input.</td>
</tr>
<tr>
<td>Assumptions:</td>
<td>• Patient is registered.</td>
</tr>
<tr>
<td>Outcomes:</td>
<td>• Permissions are changed.</td>
</tr>
</tbody>
</table>
Chapter 4. The Work

Table 4.5: User Story 5 - Essential patient information access: doctor

<table>
<thead>
<tr>
<th>ID: H5</th>
<th>User Story: Essential patient information access: doctor</th>
</tr>
</thead>
<tbody>
<tr>
<td>As a doctor, I want to access my patient’s essential medical information, so that I can get a clearer picture of their health and perform better diagnosis and treatments.</td>
<td></td>
</tr>
</tbody>
</table>

Acceptance Criteria:
- The requested information is accurately sent to the doctor.

Assumptions:
- Patient is registered;
- Doctor is registered.

Outcomes:
- Doctor receives the information.

Table 4.6: User Story 6 - Essential patient information access: patient

<table>
<thead>
<tr>
<th>ID: H6</th>
<th>User Story: Essential patient information access: patient</th>
</tr>
</thead>
<tbody>
<tr>
<td>As a patient, I want to access my essential medical information, so that I can get a clearer picture of my health.</td>
<td></td>
</tr>
</tbody>
</table>

Acceptance Criteria:
- The requested information is accurately sent to the patient.

Assumptions:
- Patient is registered.

Outcomes:
- Patient receives the information.

Table 4.7: User Story 7 - Essential patient information changes

<table>
<thead>
<tr>
<th>ID: H7</th>
<th>User Story: Essential patient information changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>As a doctor, I want to add to/update my patient’s essential medical information, so that it reflects the current status and can be used for more accurate diagnosis and treatments in the future.</td>
<td></td>
</tr>
</tbody>
</table>

Acceptance Criteria:
- Information is correctly added/updated to the blockchain.

Assumptions:
- Patient is registered;
- Doctor is registered;
- Doctor has access to the patient’s data.

Outcomes:
- The information is added/updated.

Table 4.8: User Story 8 - Medical history access: doctor

<table>
<thead>
<tr>
<th>ID: H8</th>
<th>User Story: Medical history access: doctor</th>
</tr>
</thead>
<tbody>
<tr>
<td>As a doctor, I want to access my patient’s medical history, so that I can be better informed about past medical issues and medications to perform a better diagnosis and treatments.</td>
<td></td>
</tr>
</tbody>
</table>

Acceptance Criteria:
- The requested data is accurately sent to the doctor.

Assumptions:
- Patient is registered;
- Doctor is registered;
- Doctor has access to the patient’s medical history that is being requested.

Outcomes:
- Doctor receives the data.
Table 4.9: User Story 9 - Medical history access: patient

<table>
<thead>
<tr>
<th>ID: H9</th>
<th>User Story: Medical history access: patient</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>As a patient, I want to access my medical history, so that I can be better informed about past medical issues and medications.</td>
</tr>
<tr>
<td>Acceptance Criteria:</td>
<td></td>
</tr>
<tr>
<td>• The requested data is accurately sent to the patient.</td>
<td></td>
</tr>
<tr>
<td>Assumptions:</td>
<td>Outcomes:</td>
</tr>
<tr>
<td>• Patient is registered.</td>
<td>• Patient receives the data.</td>
</tr>
</tbody>
</table>

Table 4.10: User Story 10 - Medical history changes

<table>
<thead>
<tr>
<th>ID: H10</th>
<th>User Story: Medical history changes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>As a doctor, I want to add to/update my patient’s medical history, so that it can be used for more accurate diagnosis and treatments in the future.</td>
</tr>
<tr>
<td>Acceptance Criteria:</td>
<td></td>
</tr>
<tr>
<td>• Data is correctly added/updated to the blockchain.</td>
<td></td>
</tr>
<tr>
<td>Assumptions:</td>
<td>Outcomes:</td>
</tr>
<tr>
<td>• Patient is registered;</td>
<td>• The data is added/updated.</td>
</tr>
<tr>
<td>• Doctor is registered;</td>
<td></td>
</tr>
<tr>
<td>• Doctor has access to the patient’s medical history that is being requested.</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.11: User Story 11 - Consultation history access: doctor

<table>
<thead>
<tr>
<th>ID: H11</th>
<th>User Story: Consultation history access: doctor</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>As a doctor, I want to access my patient’s consultation history, so that I can see previous visits to the doctor, the attending physician, the reason for the consultation to perform a better diagnosis and treatments.</td>
</tr>
<tr>
<td>Acceptance Criteria:</td>
<td></td>
</tr>
<tr>
<td>• The requested data is accurately sent to the doctor.</td>
<td></td>
</tr>
<tr>
<td>Assumptions:</td>
<td>Outcomes:</td>
</tr>
<tr>
<td>• Patient is registered;</td>
<td>• Doctor receives the data.</td>
</tr>
<tr>
<td>• Doctor is registered;</td>
<td></td>
</tr>
<tr>
<td>• Doctor has access to the patient’s consultation history that is being requested.</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.12: User Story 12 - Consultation history access: patient

<table>
<thead>
<tr>
<th>ID: H12</th>
<th>User Story: Consultation history access: patient</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>As a patient, I want to access my consultation history, so that I can see previous visits to the doctor, the attending physician, the reason for the consultation.</td>
</tr>
<tr>
<td>Acceptance Criteria:</td>
<td></td>
</tr>
<tr>
<td>• The requested data is accurately sent to the patient.</td>
<td></td>
</tr>
<tr>
<td>Assumptions:</td>
<td>Outcomes:</td>
</tr>
<tr>
<td>• Patient is registered.</td>
<td>• Patient receives the data.</td>
</tr>
</tbody>
</table>
### Table 4.13: User Story 13 - Consultation history changes

<table>
<thead>
<tr>
<th>ID: H13</th>
<th>User Story: Consultation history changes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>As a doctor, I want to add to my patient’s consultation history, so that it can be used for more accurate diagnostics and treatments in the future.</td>
</tr>
<tr>
<td>Acceptance Criteria:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• The data is correctly added to the blockchain.</td>
</tr>
<tr>
<td>Assumptions:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Patient is registered;</td>
</tr>
<tr>
<td></td>
<td>• Doctor is registered;</td>
</tr>
<tr>
<td></td>
<td>• Doctor has access to the patient’s consultation history that is being requested.</td>
</tr>
<tr>
<td>Outcomes:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• The data is added/updated.</td>
</tr>
</tbody>
</table>

### Table 4.14: User Story 14 - Adding analyses reports

<table>
<thead>
<tr>
<th>ID: H14</th>
<th>User Story: Adding analyses report</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>As a medical lab technician, I want to add the report of the patient’s analyses, so that it can be used for more accurate diagnostics and treatments.</td>
</tr>
<tr>
<td>Acceptance Criteria:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Report is correctly added to the blockchain.</td>
</tr>
<tr>
<td>Assumptions:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Patient is registered;</td>
</tr>
<tr>
<td></td>
<td>• Medical lab is registered.</td>
</tr>
<tr>
<td>Outcomes:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• The report is added.</td>
</tr>
</tbody>
</table>

### Table 4.15: User Story 15 – Analyses report access: doctor

<table>
<thead>
<tr>
<th>ID: H15</th>
<th>User Story: Analyses reports access: doctor</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>As a doctor, I want to access my patient’s analyses reports, so that I can use them to perform a better diagnosis and treatments.</td>
</tr>
<tr>
<td>Acceptance Criteria:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• The requested data is accurately sent to the doctor.</td>
</tr>
<tr>
<td>Assumptions:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Patient is registered;</td>
</tr>
<tr>
<td></td>
<td>• Doctor is registered;</td>
</tr>
<tr>
<td></td>
<td>• Doctor has access to the patient’s analyses report that are being requested.</td>
</tr>
<tr>
<td>Outcomes:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Doctor receives the data.</td>
</tr>
</tbody>
</table>

### Table 4.16: User Story 16 – Analyses report access: patient

<table>
<thead>
<tr>
<th>ID: H16</th>
<th>User Story: Analyses reports access: patient</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>As a patient, I want to access my analyses reports, so that I can see the results of the analyses.</td>
</tr>
<tr>
<td>Acceptance Criteria:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• The requested data is accurately sent to the patient.</td>
</tr>
<tr>
<td>Assumptions:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Patient is registered.</td>
</tr>
<tr>
<td>Outcomes:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Patient receives the data.</td>
</tr>
</tbody>
</table>
Table 4.17: User Story 17 – Analyses report access: medical lab

<table>
<thead>
<tr>
<th>ID: H17</th>
<th>User Story: Analyses reports access: medical lab</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>As a medical lab technician, I want to access analyses reports added by the lab, so that I can review them and have them available for any potential audits.</td>
</tr>
</tbody>
</table>

Acceptance Criteria:
- The requested data is accurately sent to the medical lab.

Assumptions:
- Medical lab is registered.

Outcomes:
- Medical lab receives the data.

4.5. Selected Blockchain

After analyzing the blockchain implementations described in 3.2.2., I decided to use Hyperledger Fabric. This decision was based on a set of requirements the underlying blockchain had to have, the most important of which were privacy guarantees. Since Fabric is private and with chaincode (smart contracts) it can also be made permissioned, this allows for a less complex implementation of the information access control part of the platform. Another reason behind my decision was the consensus mechanism used by Fabric, byzantine fault tolerant, because for the proposed platform there was no need for the computationally demanding proof-of-work.

During my research I also came across a post by Hyperledger indicating that they are creating a Hyperledger Healthcare Working Group for the development of blockchain solutions for healthcare. [27] This working group that also includes Accenture, the host organization for my work, and Gem, developers of one of the solutions presented above, served as indication that there is recognized value in using Hyperledger Fabric for healthcare blockchains.

Another reason behind my choice was the relatively large developer base and support around Hyperledger that was very useful for consultation when I ran into issues during development.

4.6. System Requirements

The requirements are divided into various subtypes according to the FURPS+ methodology.

4.6.1. Functionality

Table 4.18: Functionality Requirements

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Description</th>
<th>Rationale</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1</td>
<td>User registration system</td>
<td>The system must have a way for users to register and use its features.</td>
<td>In order to allow users to register, so they can use the platform, it must have an interface with all the necessary fields for registration. This interface will have to be connected to back-end logic that validates the data and if all is valid and adds the user to the system.</td>
</tr>
<tr>
<td>F2</td>
<td>Information access control</td>
<td>The system must have a mechanism for patients to manage who and what parts</td>
<td>To guarantee data privacy there must be an interface.</td>
</tr>
</tbody>
</table>
of their data can be seen and added/updated.

<table>
<thead>
<tr>
<th>F3</th>
<th>Data access</th>
<th>The system must possess the ability for doctors to look up a patient’s data.</th>
<th>That allows patients to select what people and/or institutions can access their data and which data they are allowed to access. The business logic to enforce these authorizations must also be properly implemented.</th>
</tr>
</thead>
<tbody>
<tr>
<td>F4</td>
<td>Data presentation</td>
<td>The system must include a way for the requested data to be presented to the user who requested it.</td>
<td>If a user has access to the information they requested, that information must be presented in an interface. If they do not, that fact must be presented in the same interface.</td>
</tr>
<tr>
<td>F5</td>
<td>Data update</td>
<td>The system must support the update of existing data by a doctor.</td>
<td>In the same interface where data is displayed, there must be a function to allow a doctor with access to edit and update the data presented there.</td>
</tr>
<tr>
<td>F6</td>
<td>Data insertion</td>
<td>The system must allow the addition of new patient data by a doctor or medical lab.</td>
<td>An interface has to be implemented to allow doctors or medical labs to add new data about a patient. It must also be implemented with supporting business logic to verify permission to add said data.</td>
</tr>
</tbody>
</table>

### 4.6.2. Usability

*Table 4.19: Usability Requirements*

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Description</th>
<th>Rationale</th>
</tr>
</thead>
<tbody>
<tr>
<td>U1</td>
<td>User friendly interfaces</td>
<td>The system’s interfaces shall be designed in a way the makes them easy to understand by the users.</td>
<td>In designing the interfaces there must be a special attention given to ensure they use clear language, to avoid confusion and mistakes. Moreover, the color scheme should also not cause distraction or provoke any sort of confusion in the use of the system.</td>
</tr>
<tr>
<td><strong>U2</strong></td>
<td>Consistency across the platform</td>
<td>The system shall maintain a consistent aesthetic through its multiple interfaces.</td>
<td>Consistency will be maintained across the various interfaces of the platform. This will be reflected in a common color scheme, type of language and layout. The goal is to make it easier for users to navigate and use the platform.</td>
</tr>
<tr>
<td>------</td>
<td>-----------------------------</td>
<td>-------------------------------------------------------------------------------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>U3</strong></td>
<td>Flow</td>
<td>The system shall have a logical flow of actions and interfaces to perform tasks.</td>
<td>In order to simplify user interaction, the system has to be implemented with a coherent, logical and simple flow of interactions to reach an end goal.</td>
</tr>
<tr>
<td><strong>U4</strong></td>
<td>User validation</td>
<td>The system shall support the ability of users to validate the data they want to update/add before the system validates and add it.</td>
<td>To help combat user errors that might lead to inaccurate information being stored in the system, there must be a mechanism that prompts users to check in indicate the accuracy of the data they want to update/add to the system.</td>
</tr>
<tr>
<td><strong>U5</strong></td>
<td>Undo</td>
<td>The system shall allow users to backtrack their actions.</td>
<td>In order to help users, resolve their mistakes the system will make it easy to undo certain actions.</td>
</tr>
<tr>
<td><strong>U6</strong></td>
<td>User notification</td>
<td>The system shall inform users when they try to perform actions they have no permission to, if what they are searching for doesn’t exists, if an action is not possible other reasons or if their action was successful.</td>
<td>To ensures users are always aware of the result of their actions the platform will have notifications telling them of the success or unsuccess of them. In case of success requesting information, it will just display that information. If an action was not successful the notification will be accompanied by a brief explanation.</td>
</tr>
</tbody>
</table>
4.6.3. Reliability

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Description</th>
<th>Rationale</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>Accuracy</td>
<td>All data supplied by any of the three types of user shall be accurately recorded.</td>
<td>The platform will store the information accurately to ensure users can take full advantage of the features and no misdiagnosis happen due to inaccurate data.</td>
</tr>
<tr>
<td>R2</td>
<td>Availability</td>
<td>The platform’s data shall be available for access whenever it’s needed by the users.</td>
<td>Medical data will be available when needed. This is highly important because in medical settings time critical.</td>
</tr>
<tr>
<td>R3</td>
<td>Recoverability</td>
<td>It shall be possible to recover the local copies of data each user has.</td>
<td>The system will make it possible for the recovery of the local ledgers to be recovered in case they are lost.</td>
</tr>
</tbody>
</table>

4.6.4. Performance

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Description</th>
<th>Rationale</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>Response Time</td>
<td>The system shall respond to user commands in at most 1 second.</td>
<td>The system has to be designed to be responsive and have a reaction time that allows users to keep their string of thought unbroken.</td>
</tr>
</tbody>
</table>

4.6.5. Supportability

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Description</th>
<th>Rationale</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>Testability</td>
<td>The system shall be developed in a way that allows tests to be easily performed.</td>
<td>Given the context of the development behind the development of the platform, it will be designed in a way that easily supports running various types of tests to it.</td>
</tr>
<tr>
<td>S2</td>
<td>Extensibility</td>
<td>The system shall be design in a way that allows for more functions, data and users types to be added in the future.</td>
<td>Since the platform could be a used as a possible basis a bigger system in the future, it will be developed in a way that supports the addition of new features.</td>
</tr>
<tr>
<td>S3</td>
<td>Modularity</td>
<td>The system shall be design in modules.</td>
<td>The multiple features of the system will be implemented</td>
</tr>
</tbody>
</table>
in modules to allow for an ease in testing, extensibility and reduction of redundant code.

4.6.6. Design Constraints

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Description</th>
<th>Rationale</th>
</tr>
</thead>
<tbody>
<tr>
<td>DC1</td>
<td>Blockchain</td>
<td>The system shall be implemented using blockchain.</td>
<td>The use of blockchain will impose certain architectural and design ideas on the platform.</td>
</tr>
<tr>
<td>DC2</td>
<td>Data Privacy</td>
<td>The system shall implement mechanisms that allow the privacy and confidentiality of the data.</td>
<td>By requiring data to be kept private and enforcing ownership by the patients, who will also manage access to it, the blockchain used will have to be both private and permissioned.</td>
</tr>
<tr>
<td>DC3</td>
<td>Multiple user types</td>
<td>The system shall support different types of users.</td>
<td>The system will have to be designed to support different types of users but also the different actions and roles they will have.</td>
</tr>
<tr>
<td>DC4</td>
<td>Multiple data types</td>
<td>The system shall support different types of data.</td>
<td>The system will have to be designed to support all the different types with different fields and sets of information. These will also have to support the data privacy and access control.</td>
</tr>
</tbody>
</table>

4.6.7. Implementation Constraints

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Description</th>
<th>Rationale</th>
</tr>
</thead>
<tbody>
<tr>
<td>ImC1</td>
<td>Hyperledger Fabric</td>
<td>The system shall use Hyperledger Fabric.</td>
<td>The chosen blockchain will require the use of specific programming languages, impact the architecture and structure of the system.</td>
</tr>
<tr>
<td>ImC2</td>
<td>Chaincode languages</td>
<td>Chaincode shall be implemented using one of the supported languages.</td>
<td>Chaincode, smart contracts in Hyperledger Fabric, must be written in Go, Java or Node.js.</td>
</tr>
</tbody>
</table>
4.6.8. Interface Constraints

There are no applicable interface constraints for this platform, because it will not communicate with any external systems. Any data needed for the system to work from the start will be supplied directly during development and testing.

4.6.9. Physical Constraints

There are no applicable physical constraints for this platform. It will run on an Ubuntu virtual machine installed on a Windows laptop.

4.7. The Platform

The platform will store four types of medical data:

- Essential Medical Information: the most current medical information, it contains the patient’s height, weight, blood type, blood pressure, active allergies and medical conditions and medications being taken;
Chapter 4. The Work

- Medical History: a set of lists storing all the allergies, medical conditions and medications the patient ever had;
- Consultation History: record of the patient’s consultations;
- Analyses Reports: record of every analyses report concerning the patient.

Each of these data types as a chaincode associated with it. It is in those chaincodes that they are defined, and where the business logic and respective access control is implemented. The organization and contents of said chaincode is explained in more detail in the following sections of this chapter.

While in the back-end the data is defined directly in the chaincode, in the front-end each type has its own module with various components and a service to handle user interactions, requests and responses to those requests.

Besides the four main medical data types described above, there are other types in the platform:
- Allergy: contains a name, symptoms, date of detection, cause, status and a list of medications to treat it;
- Medical Condition: can represent illnesses, injuries or mental issues. Is composed of type, name, symptoms, status, diagnosis date, date of cure and a list of medications to treat it;
- Medication: has the following attributes, name, dosage, prescription date, end of treatment, condition it was prescribed for (allergy or medical condition);
- Consultation: comprised of the specialty, attending physician, ID of the patient, date it was conducted, reason and its conclusions;
- Analyses Report: contains the ID of the patient and the medical lab, type of analyses, results of the analyses in the form of a PDF.

The non-medical data of the users is also stored in the blockchain. This data has the same back-end and front-end organization of the medical data. There are three types of users with different functions they can perform in the platform:
- Patients: can access all of their medical data, search it and control access to it. They have an ID, password, name, e-mail, emergency contact, list for the authorized doctors with their permission types for the various medical data types and a similar one for the medical organizations;
- Doctors: have the ability to start consultations, add them to the blockchain, update or add medical data during them and search the patient’s data. All of this is dependent on the doctor’s permissions or, in case he has none, the permissions of the organization they work at. Doctors have an ID, password, name, e-mail, specialty and name of the organization where they work;
- Medical Laboratories: are able to add analyses reports for patients and search reports they added to the blockchain. Each medical lab has an ID, password, name and e-mail.

4.8. Platform Architecture

In figure 4.3 there is a representation of the platform’s architecture. It shows the flow of a request form the client application to the ledger.
4.9. Blockchain Network Setup

With the platform, requirements and architecture defined the next stage of work was the setup of the blockchain network. Before it could be done, the work environment needed to be created. This meant installing an Ubuntu virtual machine with all the required programs, tools, frameworks and programming languages in the correct versions. Once everything was properly installed the most recent and stable version of Hyperledger Fabric, at the time 1.4.0, was downloaded from Git along with the appropriate Docker images. The Fabric Certificate Authority and samples, the latter of which contains the source code for the tutorials, were also downloaded. With all of this completed and after the tutorials were done, to learn more about how to develop in Fabric, the setup began.

4.9.1. Hyperledger Fabric Concepts

Hyperledger Fabric has some specific concepts that don’t exist in other blockchain implementations and in also refers to others by different names. In this section the most important and relevant of those concepts will be briefly explained.

4.9.1.1. Chaincode

This term was mentioned a few times before in this report. Chaincode is the name Fabric gives to smart contracts. There is no difference between the definition of smart contract given in the State of the Art chapter and the one used by Hyperledger Fabric, only the name is different. There can be multiple chaincodes in the same network. The practice is to divide the functionalities into multiple chaincodes, similar to modules in Java.

4.9.1.2. Nodes

Fabric divides nodes into three types with different roles in the network:

- Client: node that submits transaction proposals to the network for the user. It must be connected to a peer for the communication to be possible;
Peer: node that commits the transactions, if validated by the consensus mechanism, they maintain a copy of the ledger and can have multiple chaincodes installed. If a peer is also involved in the process of validating a transaction it is also called an endorser;

Orderer: node the runs the communication service that provides delivery guarantees. One of these guarantees is the atomic delivery of all the messages in the channel.

### 4.9.1.3. Channel

A channel provides a communication medium between a sub-group of nodes, though it can include all the members of a network. It serves to maintain transactions private to all but the parties with a need-to-know. Each channel as its own shared ledger, meaning that peers that belong to multiple channels have multiple ledgers. This too serves to maintain privacy.

### 4.9.1.4. Organizations

In Fabric nodes belong to organizations. These organizations manage their respective members under the same membership service provider, MSP. In a business sense, organizations can be companies that belong to the network.

### 4.9.1.5. Membership Service Provider

An MSP is used to identify the trusted members of a Fabric organization. It serves to identify and validate the certificates, used for identity of each member and to retrace them to the trusted root certificate authority that issued them. As alluded to before each organization as its own MSP.

### 4.9.1.6. Certificate Authority

Trusted entities that issue valid certificates that nodes can use to authenticate themselves in the network. Hyperledger Fabric has its own implementation of a CA, Fabric CA.

### 4.9.1.7. State Database

Besides the permanent record of every transaction, in Fabric the current state is stored in a state database. The data here is stored in a key-value format to facilitate queries and other operations.

### 4.9.2. Platform Network

The source code of the tutorials was used as a basis for network setup. The network has three organizations, one for the patients Orgpatient, one for the doctors Orgdoctor and one for the medical laboratories Orgmedlab. Each of these organizations has its own MSP and CA. For Orgpatient these are OrgpatientMSP and ca_orgpatient. The doctor and med lab organizations’ MSP and certificate authorities are named following the same convention, OrgdoctorMSP, ca_orgdoctor, OrgmedlabMSP and ca_orgmedlab, respectively.

Since access to the data was going be restricted via the access control, the small scale of the network and the unnecessary extra strain in the resources, i.e. the CPU, the network only has one channel were all three organizations communicate, threeorgschannel. It is through this channel that all communications in the blockchain network are conducted. Due to this there is only one orderer in the network, named simply orderer.

Every organization has two peers each. These are named peer0.orgpatient and peer1.orgpatient for the patient’s organization, with the peers of the other organizations having the same names only changing the org part for their respective organization ids. Only the peer0 of each organization is an endorser. The web-app is the client node. When a user logs in, the certificate indicates to the network
which MSP it belongs to, thus identifying the user’s organization. Communication between the client node and the peer, and by extension the network, is made using a server running locally.

The data types and functionalities were divided into five chaincodes:

- analyses-report;
- consultation-history;
- medical-history;
- medical-information;
- users.

These will be explained in detail in the next section of this chapter. All five are installed in each of the peers of every organization.

The state database used is CouchDB. This option was chosen because it supports not only the standard key-value queries available in LevelDB, the default state database, but also rich queries. These are more complex queries to the chaincode that search for data based on attributes not the key. An analogy with traditional relational databases would be search for every entry on a table with a certain value in one of the columns.

Below is an example of what a peer of the network looks like and a diagram of the network as a whole. In the network diagram, for ease of representation the chaincodes and ledger of each peer are not shown, and neither is the server. This last one is only represented with an arrow going from the clients to the channel.

Figure 4.4: Peer

Figure 4.5: Blockchain Network
4.10. Platform Development

After setting up the network, the work moved to the development of the chaincode with the functionalities identified, implement methods in the contract classes, the front-end that allows users to take advantages of said functionalities and the server to send requests to the proper smart contract and the response back to the user.

This section is divided into seven sub-sections, one for each chaincode, one for the access control and one for shared components in the front-end. The chaincode sub-sections explain not just the chaincode structure, methods and logic, but also the front-end modules components and services and the server that directly communicates with that chaincode. In the access control sub-section, the implementation across every chaincode is explained. There will be no reference to access control in the chaincode sub-sections to avoid repetition and focus them on the functionalities of the platform. The shared components section presents the components and data interfaces used in multiple modules of the web application.

4.10.1. Medical Information

A medical information is composed of the following attributes:

- medical ID: unique identifier of the patient who owns the information, it is the key and, as such, is used to search for the information;
- height: height of the patient;
- weight: weight of the patient;
- blood type: blood type of the patient;
- blood pressure: blood pressure of the patient;
- allergies: list of allergy objects with active status;
- current medications: list of medication objects with an end of treatment date greater than the current date;
- current medical conditions: list of medical condition objects with active status.

The first five attributes are in the form of strings, because the Fabric SDK for Node.js does not support transactions with non-string attributes. The object is converted to string format and placed inside a buffer during communication between server and chaincode.

When a new user registers in the platform, their medical information object is created. During the registration, they can input the height, weight and blood type, should they know any of those. If nothing is submitted in that step of registering, those attributes will be initialized as empty strings, along with the blood pressure while the lists are initialized empty.

During consultations doctors can update the medical information, this can be directly in the case of the height, weight, blood pressure and type, or indirectly in case of allergies, medical conditions or medications as they would be added to the medical history chaincode and depending on their status or end of treatment date that chaincode would invoke the medical information chaincode to perform the update. Updates to the medical history can also trigger a removal from the medical information of an allergy, medical condition or medication, in case they stop being active or currently being taken.

The only other functionality of the medical information is accessing it, this can be done by the owner or any doctor.

4.10.1.1. Chaincode

The medical information chaincode has a MedicalInformation class defining the medical information with the above-mentioned attributes and methods to get attributes, set new values when they are updated, add/update a new allergy, medication or medical condition guaranteeing no repetitions and
remove allergies, medication and medical conditions when they are no longer active or being taken. There are also methods to serialize, deserialize the object as well as methods to add the object into a new buffer or retrieve a medical information object from a buffer. These last four methods are needed to store the data in the ledger, send it to the client and convert them into the proper type, after being received from the client.

Besides a class for the medical information there are also three classes defining allergies, medical conditions and medications, with the attributes mentioned above in this chapter. These classes also possess the same methods for serialization, deserialization and buffer interaction as the MedicalInformation.

Just like it was explained in the section 2.3. Methodology of chapter 2, the medical information smart contract also contains a MedicalInformationList class that maintains a list of every medical information and sends requests from the contract class to the StateList. The StateList class serves to provide a virtual container for the objects as a set of ledger states accessible by their key. It contains methods to add, update and get the state, these are properly serialized or deserialized by the methods.

The MedicalInformationContract class is where the functionalities are actually implemented. This class’ methods receive the requests from the server or other chaincodes and perform the required actions. It has the following methods:

- **registerMedicalInformation:** receives the medical ID, height, weight, blood type and pressure, the last four can be empty strings. Verifies if there is no medical information for that patient already created and, if not, creates a new object and calls the list class to add it to the ledger, finally it sends the newly created object back to the client;

- **updateMedicalInformation:** receives the medical ID, information field (value indicating which attribute is being changed) and the new value. It retrieves the medical information with the medical ID, updates the attribute with the new value. In case this new value is an allergy, medical condition or medication, it first parses it from a string into the correct object. Finally, it calls the list class to perform the update and sends the updated medical information to the caller (client or another chaincode);

- **removeMedicalInformation:** receives the medical ID, information field (only applies to allergies, medications and medical conditions) and the name of what is to be removed. The method is very similar to the update, retrieves the medical information object, calls the proper remove method from MedicalInformation, calls the update method of the list class and sends the medical information object to the caller.;

- **getMedicalInformationbyMedicalID:** receives the medical ID and sends the medical information object to the client. It also performs an intermediate verification to check that no medications are being stored with an end of treatment date that have already passed. If so these are removed, and the update method of the list class is called for the update, prior to sending the object to the client.

All the previously described methods throw errors back to the client when no medical information is found.

### 4.10.1.2. Front-end

The medical information in the web-app has its own service, module, component and data interface.

The interface has the same attributes as the chaincode class. There are also interfaces for allergies, medications and medical conditions with interfaces and components for creation, display and update, but these are not specific for the medical information, so they will be explained later.

The medical information service has four methods, one for each of the chaincode contract class methods described above. It sends the client’s requests to the chaincode via HTTP POST or GET
requests to the server with the proper attributes, along with the ID and type of the user making the request. The latter two are used by the server to retrieve the proper credentials of the user to authenticate it in the network and be used for the access control by the chaincode.

The medical information module only has one component, the details. This component is responsible for presenting the medical information. When opened it calls the get method of the medical information service to retrieve the medical information to be displayed. It also calls the get patient method of the user service to retrieve the patient’s name to display along with the ID. The allergies, medications and medical conditions are presented as a list of their names, that when clicked open a dialog with all the details. The details component is present for the patient and for the doctor, though for the latter it is on a tab of the consultation component.

Figure 4.6 is a screenshot of details component:

![Figure 4.6: Medical Information Details Component](image)

### 4.10.1.3. Server

The server receives the requests from the service and sends them to proper method of the chaincode’s contract class. It has four, just like the contract class and the service. These methods will use the user ID and type to retrieve proper credentials and, if they exist, make the request. They then receive the response from the chaincode and send it to the front-end.

### 4.10.2. Analyses Reports

The analyses report has five attributes:

- patient ID: medical ID of the patient who owns the report;
- laboratory ID: unique identifier of the laboratory that performed the tests, and submitted the report to the blockchain;
- date conducted: date when the tests were conducted;
- type: numerical representation of the type of analyses the report refers to, currently there are three types, 1 – x-ray, 2 – blood test and 3 – MRI;
- results: PDF of the report stored as a base 64 string.

The analyses report object has a composite key created by joining the patient ID, laboratory ID and the date conducted.

Medical laboratories registered in the platform can create new analyses report objects for existing patients by submitting a create request with all the first four attributes filled. The results are added in a second request. The medical lab user sees this as only one interaction, a create form that requires the four fields and the PDF to be inputted. The addition of the results is done in a separate method due to the need to convert it into a base 64 string.
Users can also make queries to obtain analyses reports. These can be to get a specific one, by inputting all the attributes that form a key, or requests for reports of a certain patient or type (for the same patient).

4.10.2.1. Chaincode

The analyses report chaincode shares many similarities with the medical information one. The analyses report is defined in the AnalysesReport class. This class defines the object with the attributes described above, all strings for previously mentioned reasons. It also has methods to return each of the attributes and a method to add the string version of the report to the object. Much like the MedicalInformation class this one also has serialization, deserialization methods, as well as, methods to retrieve an analyses report from a buffer and create a buffer with a string version of an analyses report object. Since the analyses report key is a composite, its class possesses a method to create the key using the different attributes and another to retrieve all three from a key.

The AnalysesReportList class maintains a list of all analyses report object in the world state and offers methods to create a new one, update it with the results, and for the multiple available queries. This class sends the requests to the StateList, much like the list class of the medical information chaincode.

The analyses report chaincode’s StateList class serves the same purpose as the one described in the Medical Information sub-section. The methods to add a new object, update it and retrieve it using the full key are also nearly identical. The methods to get all the reports of a patient and all the patient’s reports of a certain type, however, take full advantage of the rich query capability of the CouchDB state database. These use selectors to identify all the objects, in the form of states, with patient ID or patient ID and type attributes that match the requested values. For this type of queries to work effectively this chaincode also possesses two indexes, one for each of the rich queries. The indexes identify the JSON fields that are to be searched in the query, the patient ID or the patient ID and the type.

The AnalysesReportContract class defines the methods that receive the request from the server and act upon them. These methods are:

- createAnalysesReport: receives the patient ID, laboratory ID, date conducted and type. First it verifies that the report has not been created before and, if not, creates a new analyses report object and calls the list class to add it to the ledger. Finally, it sends the new object to the client;
- addResults: receives the three key components along with the base 64 string of the report file. Requests the object from the list class, adds the results and calls the update method of the AnalysesReportList. After the analyses report object is correctly updated in the ledger, it sends it to the client;
- getAnalysesReport: receives the patient, lab ID and date conducted and calls the AnalysesReportList method getAnalysesReport to retrieve the requested object. Once it has it, sends it to the client. In case the report does not exist, it sends an error to the client;
- getAnalysesReportByPatient: receives the patient ID. Then it calls the appropriate list method and converts the result, a list of analyses report objects, into a string using JSON.stringify. Lastly, it sends this to the client;
- getAnalysesReportByType: receives the patient ID and the type. Performs a nearly identical set of actions as the previous method but calls a different get method from the list class.

4.10.2.2. Front-end

The analyses report object in the front-end is defined by an interface with the same attributes and types as those defined in the chaincode.
The analyses report service has five methods, these are responsible for sending to the server the create new report, add results and different query requests. Much like in the case of the medical information this services’ methods also send to the server the ID and type of user making the request.

The analyses report module has four components, one for creating a new report, two to display the detailed report (one in a full page, the other in a dialog), and the last one for searching. The create component has a simple form that requires the medical lab user to input the patient ID, date conducted, the type and the file. After all of these have been correctly filled it calls the service to create the new report on the blockchain. When it receives the response and verifies that the report was correctly added, it sends a call to the service method that makes the add results request. Finally, after verifying that the results were properly added, it sends the user into the details page. Both details components are nearly identical, changing only how the analyses report is displayed. They present the names of the patient and laboratory instead of the ID, this is achieved by calling the get patient and get lab methods of the user service, the date conducted, the type of the analyses, in text form converting the numerical representation, and the results. The results are presented as a button, that when clicked invokes a method in the component that converts the string back into a PDF and opens it in a new tab. The reason for having two different components that perform that same actions and display the data in the same way is due to what component calls them. The full-page component is called after creating the report with the results and when searching for a specific one, while the dialog is opened in the results of searches by patient or patient and type. The search component, shared by all three user types, possesses a form for the user to input the fields for the search. If the user is a medical lab, the medical lab field will already display the ID, if it is a patient the patient ID will be filled and if it is a doctor, the patient ID will have the ID of the patient the consultation refers to. After the search button is pressed, the component selects the correct service method to call based on what fields have input. The results are displayed on a table and, to view the detailed report the user simply has to click the row for the dialog to be opened.

Below are screenshots of all four components:

![Figure 4.7: Analyses Report Create Component](image1)

![Figure 4.8: Analyses Report Details Page Component](image2)
Both figures 4.9 and 4.10 were taken from the perspective of patient with ID 123456, the search performed was a search by patient ID.

### 4.10.2.3. Server

Identically to the medical information, the server receives requests from the web application. Its methods send those requests to the proper methods on the chaincode and return the chaincode’s response to the application.

The only major difference is that the, add results method receives the PDF file and converts it into a base 64 string. It is this string that is sent along with the other values received from the client to the chaincode.

### 4.10.3. Medical History

A medical history has four attributes:

- medical ID: medical ID of the patient who owns the medical history;
- allergies: list of allergy objects, representing all allergies active or inactive the patient has or had;
- medications: list of medication objects, representing all medications prescribed to the patient;
- medical conditions: list of medical condition objects, representing all medical conditions active or inactive the patient has or had.

Like in the medical information the key of a medical history object is the medical ID. This does not create conflicts because all keys have the chaincode name added to the beginning by the create key method of the Node.js SDK.

A medical history is created with empty lists of allergies, medical conditions and medications when a patient signs-up in the platform, just like the medical information.

Data is added to the medical history during consultations. Doctors can add new allergies, medical conditions and medications or update them, changing the status or other attributes.

Both doctors and patients can search the medical history, by type (allergy, medication or medical condition), type (allergy or medical condition) and status (active or inactive), end of treatment date of medications, type of medical condition (illness, injure or mental illness) or the full history.

#### 4.10.3.1. Chaincode

This chaincode possesses the same classes for allergies, medications and medical conditions as the medical information chaincode. The medical history is defined by the MedicalHistory class. This class has all four attributes described above, methods to get the lists of allergies, medications or medical
conditions and methods to add/update allergies, medications and medical conditions. Just like the MedicalInformation and AnalysesReport classes, MedicalHistory also possesses methods to read a medical history object from a buffer, to create a buffer and insert a medical history object into it and to serialize and deserialize medical history objects.

The MedicalHistoryList class maintains a list of every medical history object. It possesses methods for creation and update of the medical history as well as methods for all the possible queries. This class’ methods call the StateList class’ methods.

The StateList class, much like the ones on other chaincodes, maintains a virtual container with the medical history objects as a set of ledger states. The add, update methods are very similar to the ones of the other StateList classes. It has five get methods, one for each of the possible searches. Unlike in the analyses report chaincode, this StateList does not make use of selectors and indexes for the more complex searches, instead it retrieves the full medical history from the ledger and then returns the lists or sub-sets of the lists that satisfy the search criteria.

Just as in the other chaincodes, the MedicalHistoryContract class implements methods for the creation, addition/update of data and for the searches. All the search methods are almost identical, changing only in the parameters they receive and what MedicalHistoryList methods they call, as such, only one is described in detail while the others will include only the name, parameters, and list class method called. The methods are:

- **createMedicalHistory:** receives the medical ID of the patient. Checks to make sure the medical history was not previously created and, if not, creates it and calls the MedicalHistoryList class method to add it to the ledger. Lastly, it sends the newly created object to the client;
- **updateMedicalHistory:** receives the medical ID, history field (type of data being added/updated), the new value and the current date (date at time of update on the client side). After retrieving the medical history object from the ledger, it parses the new value and creates an allergy, medical condition or medication object and calls the correct MedicalHistory method to add the new value. For allergies and medical conditions, if the status is active, it invokes the updateMedicalInformation method of the MedicalInformation-Contract class of medical information chaincode, to update it. If the status is inactive, it invokes the removeMedicalInformation method to remove it from the medical information. In case it is a medication and the end of treatment date as passed, i.e. is before the current date, it invokes the removeMedicalInformation, if not it calls the updateMedicalInformation. Finally, it calls the list class update method to add the update history object to the ledger and then sends it to the client;
- **getMedicalHistoryByMedicalID:** receives the medical ID. Calls the getFullMedicalHistory method of the MedicalHistoryList class and sends the returned value to the client;
- **getMedicalHistoryByType:** receives the medical ID and type. Calls the proper list class method and returns the result in a buffer, just like the other methods, to the client;
- **getMedicalHistoryByTreatmentDate:** receives the medical ID and date. Calls list class method to get all medication with end of treatment date prior to date. It then sends the result back to the patient;
- **getMedicalHistoryByTypeAndStatus:** receives the medical ID, type and status. It will call the list class method to retrieve all allergies or medical conditions with the requested status. Lastly, it returns the resulting list back to the client;
- **getMedicalHistoryByMedicalConditionType:** receives the medical ID and the type. Calls the list class method to get all medical conditions of the requested type. Sends the resulting list to the client.
Just like in other chaincode contract classes this one’s methods all throw errors back to the client in case the medical history object does not exist.

### 4.10.3.2. Front-end

Similarly, to the other data types, the medical history is defined in the front-end by and interface. As previously alluded to the allergies, medical conditions and medications their own interfaces and components that will be described in another sub-section.

The medical history service has methods for creating, adding/updating and the multiple different searches. These methods all send the requests to the server with the needed parameters along with the ID and type of the user making the requests for authentication proposes.

The medical history module has two components, one for the details and the other for searching. The details component presents all allergies, medical conditions and medications on a list, regardless of the status or end of treatment date. When the names are clicked it opens the details dialog of that type of medical data. The search component is very similar to the search component of the analyses report, having a form where the user inputs the parameters and the business logic identifies the type of search and calls the proper service method. The results are presented on a table with the rows being buttons that when pressed open a dialog with a detailed view of the allergy, medication or medical condition. These components are available for both patients and doctors. The search component is accessible via a tab in the details component. The addition of medical history is done via the consultation component so there is no specific component for it. Updates to data already in the medical history is done via the dialog with details of the data, that allows it if the user is a doctor.

Below are screenshots of both components from the perspective of a patient:

![Medical History Details Component](image1.png)  
![Medical History Search Component](image2.png)

### 4.10.3.3. Server

The server methods for the medical history simply use the ID and type of the user making the request to select their credentials and, if found, makes the request to the correct method of the chaincode’s contract class, and sends the results to the client.

### 4.10.4. Consultation History

A consultation is composed of the following attributes:

- medical ID: ID of the patient who owns the consultation and it refers to;
- specialty: numeric representation of the consultation’s specialty, 1 – general medicine, 2 – dermatology, 3 – cardiology, 4 – ophthalmology;
- attending physician: ID of the attending physician of the consultation;
- date: date of the consultation;
- reason: reason for the consultation;
• conclusions: summary of the consultation, observations and tests performed, allergies or medical conditions diagnosed or treated (status changed to inactive), prescribed medications, etc.

In both back-end and front-end a consultation is called a ConsultationHistory object. While these objects only refer to one consultation each, the set of all of these for the same patient form the consultation history. Consultation history objects have a composite key formed by the medical ID, attending physician and date.

Doctors can create new consultations during which they can update the medical information and medical history.

Both doctors and patient can search the consultation history by specialty, date, attending physician or get the full history.

4.10.4.1. Chaincode

As mentioned above a consultation is defined by the ConsultationHistory class with all the attributes in string form. This class possesses methods that return every attribute, methods to serialize, deserialize, read an object from a buffer, create a buffer and add a consultation history to it and to create the key and get the individual parts from one.

The ConsultationHistoryList class keeps a list of all the consultation history objects and has methods that receive requests from the contract class and sends them to the proper StateList methods.

The StateList class has the same purpose has the ones in the other chaincodes. It has methods to add, update and get a consultation history object, very similar to the ones mentioned above. It also has methods for the more advanced searches that make use of selectors. This chaincode has for indexes, one for the getting all objects of with the same medical ID, medical ID and attending physician, medical ID and specialty or medical ID and date.

The ConsultationHistoryContract class has methods to implement all the functionalities of the consultation history:

• createConsultationHistory: receives the medical ID, specialty, attending physician, date, reason and conclusion. It verifies that the consultation was not previously added to the ledger and, if not, creates a new consultation history object and calls the list class method to add it. Finally, it sends the new object to the client;

• updateConsultationHistory: receives the medical ID, change field (height, weight, blood type, blood pressure, allergies, medications or medical conditions) and the new value. Despite the name this method does not actually update any consultation history object, instead, depending on the change field it invokes either the medical information or medical history chaincode to add/update the new value;

• getConsultationHistory: receives the medical ID, attending physician and the date. Calls the getConsultationHistory method of the ConsultationHistoryList class and returns the consultation history object to the client;

• getConsultationHistoryByPatient: receives the medical ID. Calls the list class method that returns a list of all consultation history objects for the patient. Lastly, it converts the list into a string and sends it to the client inside a buffer, just like all the other methods of this class and the other contract classes in the various chaincodes;

• getConsultationHistoryBySpecialty: receives the medical ID and the specialty. This method is just like the one described above but it calls a different list class method and returns a list of all consultation a patient had of the requested specialty;

• getConsultationHistoryByAttendingPhysician: receives the medical ID and attending physician. Calls the getConsultationHistoryByAttendingPhysician method of the
ConsultationHistoryList class and sends to the client a list of every consultation a patient has had with that specific physician;

- getConsultationHistoryByDate: receives the medical ID and date. Calls the list class to get a list of every consultation the patient has had in a certain date and returns it to the client.

This class throws errors back to the client in case the consultation that their trying to create already exist, when the specific consultation history object does not exist or when the update failed.

4.10.4.2. Front-end

The consultation history is defined by an interface in the front-end. Its module makes use the allergy, medication and medical condition components and data interfaces shared by other modules.

The consultation history service has methods to send every request to the server with the proper parameters along with the ID and type of the user making it.

The consultation history module has five components. The details component controls a tab view that has the create component, the search component, the medical information details component, the medical history details component, with the search tab, and the analyses report search component. The consultation history search component has a search form for users to perform all four types of searches. Just like previous search components, it calls the correct service method for the search depending on what form fields have been filled out. The results are displayed in a table and each row when clicked opens a dialog with the object’s details. The consultation history dialog presents a consultation history object with all of its details. Both the dialog and the search components are shared by patients and doctors. The create component calls the user service to get the patient and doctor names and other relevant information to start the consultation and display in the view. It has a form to input all of a consultation’s attributes and calls the service create method to register it into the ledger when all have been filled out and the doctors clicks the create button. It possesses another form for updating the medical information. And finally, a section for a doctor to add allergies, medical conditions and medications. Just like for the medical information and history, the added allergies, medical conditions and medications are presented as a list. When their name is clicked a dialog with the details is opened and allows doctors to update the data. This last section and the medical information form call the consultation history update method of the service. The final component is the default consultation history component that is used to start a consultation. It has a simple form where the doctor inputs the patient’s medical ID and verifies if he has permission to be the attending physician for that patient, if so, it navigates to the details component with the create tab open.

Below are screenshots of the default, details, create, search and dialog from the doctor’s perspective:

![Figure 4.13: Consultation History Default Component](image)

![Figure 4.14: Consultation History Details and Create Component](image)
4.10.4.3. Server

The server methods for the consultation history simply receive the requests from the web application in the form of HTTP POST or GET and, after getting the user’s credentials using the ID and type, send the requests to the chaincode’s contract class. Lastly, they return the chaincode’s response to the client.

4.10.5. Users

The platform has three different types of users, the patients, doctors and med labs. Each of these as its own attributes. The patient’s attributes are:

- type: 1, indication that it is a patient;
- ID: patient’s medical ID;
- pwd: hash version of the patient’s password;
- name: patient’s name;
- email: patient’s email;
- emergency contact: patient’s emergency contact;
- authorized doctors: list of permissions of the doctors authorized to access the patient’s medical data and the levels of access granted;
- authorized organizations: list of permissions of the organizations (hospitals) authorized to access the patient’s medical data and the levels of access granted.

The doctor has the following attributes:

- type: 2, indicating that the user is a doctor;
- ID: doctor’s identification;
- pwd: hash version of the doctor’s password;
- name: name of the doctor;
- email: email of the doctor;
- org name: name of the organization (hospital) the doctor works at;
- specialty: doctor’s specialty.

A med lab has five attributes:

- type: 3, indicating that the user as type med lab;
- ID: ID of the med lab;
- pwd: hashed version of the med lab’s password;
- name: medical laboratory’s name;
- email: lab’s email address.
A permission is a specific object, with the following attributes:

- **ID**: doctor’s ID or organization’s name, depending on whether it refers to a doctor or organization;
- **doctor name**: name of the doctor or empty if it refers to an organization;
- **authorized consultation type**: 0 – none; 1 – general medicine; 2 – dermatology; 3 – cardiology; 4 – ophthalmology; 5 – all;
- **authorized analyses report type**: 0 – none; 1 – x-ray; 2 – blood test; 3 – MRI; 4 – all;
- **authorized medical history**: 1 – active; 2 – all.

For each of the three type the respective key is a composite of the type as a word and the ID, for example a patient’s key would be PatientXXXXXX, where XXXXX is the ID.

Besides registering to use the platform and requesting their own user object, every type can also update some of its attributes, request other users’ data. In the case of request of data of another user, the chaincode will create a new object without the password, the only private attribute of a user. Doctors and med labs can only update their email address, while patient can update their email and emergency contact. Patients also have the ability to update their authorization lists, adding new entries, updating existing ones or removing them.

### 4.10.5.1. Chaincode

Due to the many shared attributes between the three user types and taking advantage of Node.js absence of types, there is only one object, the user defined in the User class. The user class has methods to get every attribute of each of the user types and update the email and emergency contact, the latter of which is only usable for patients. There are also methods to add/update the authorizations and to remove them. Due to this class representing three different user types, it has three different create methods, one for each. There are also methods to serialize, deserialize, read user from buffer and create buffer and insert a user into it, these work like the ones described in the other chaincodes.

The UserList class keeps a list of every user and has methods that receive requests from the contract class and call the appropriate StateList methods that interact with the ledger.

The StateList class has the same purpose as the ones in the other chaincodes. The add, get and update methods use the type attribute and a set of ‘if’ conditionals to perform the actions specific to each user type. There is also a method that uses a selector and index to retrieve every user of a certain type from the ledger. This is only used to get every doctor and from them every organization for the add permissions components in the front-end.

The UserContract class has methods that implement every available functionality for the users. For registering a new user and updating an existing one, it has one method for each user type. However, since the methods for each user type are very similar to one another, only one is described in detail, with the others being presented with their parameters and differences. The methods are:

- **registerPatient**: receives the ID, pwd, name, email and emergency contact. It starts by checking if that patient is not already registered and, if not, creates a new user object of type patient and adds it into the ledger by calling the addUser method of the list class. Finally, it sends the newly created object to the client;
- **registerDoctor**: receives the ID, pwd, name, email, org name and specialty. It works just like the registerPatient but for doctors;
- **registerMedLab**: receives the ID, pwd, name and email. Besides creating a user object with type med lab (3), there is no difference between this method and registerPatient;
- **getUser**: receives the type and ID. Calls the UserList method getUser to retrieve the user of the requested type with the requested ID. If the client is different from the user being retrieved it removes the password before returning the user object;
Chapter 4. The Work

- updatePatient: receives the ID, value field and value. It retrieves the patient from the ledger using the list class. Uses the value field to check if the client wants to update the email or emergency contact and updates the proper attribute with the value. It then calls the updateUser method of the list class and sends the updated object to the client;
- updateDoctor: receives the ID, value field and value. Works like the updatePatient method but for doctors and can only update the email;
- updateMedLab: receives the ID, value field and value. The only different between this method and the ones above is that this one updates medical laboratories;
- manageAuthorizations: receives the ID, type, user ID, doctor name, authorized consultation type, authorized analyses report type and authorized medical history. It retrieves the patient from the ledger by calling the list class using the ID. It uses the action, 1 for add/update and 2 for remove, in conjunction with the type, 1 for doctors and 2 for organizations, to call the correct User class method. If it is to set authorization for a doctor, it creates a permission with the user ID, the doctor name and the authorizations. On the other hand, if it is to set authorizations for an organization, the user ID parameter actually has the organization’s name, and it creates a permission with the user ID and the authorizations. All permissions are created by the proper set authorization methods of the User class. After all of this is done, it updates the patient by calls the updateUser method of the UserList class and returns the updated patient to the client;
- getUsersByType: receives the type. This method calls the similarly named method of the list class to get all users of a certain type. It then removes their passwords and sends the resulting list in string form inside of a buffer to the client. While this method can be used to get every user of every type, it is only used to get the list of doctors for the permissions components.

This classes’ methods send errors back to the client when the requested user does not exist.

4.10.5.2. Front-end

Since TypeScript is a typed language, instead of one interface for all user types, there is one interface for each of the user types and one for the permissions.

The users’ service has methods that receive requests from the components and send them in the form of an HTTP request to the server and then send its response back to the component. To get every doctor and organization for the permissions components it has a method called getDoctors. This method is the one that makes the requests to the getUsersByType method, always with type equal to two. Just like in every other service, this one’s methods also receive the ID and type of the user making the request and send them along with the required parameters to the server for user authentication.

The users’ module as a login component that has a form with fields for the type, ID and password for the login. Its view hides the password when it is written. After the user clicks the login button, it calls the service to get the proper user and if it does not exist or the password is incorrect, it opens a dialog with an error message. If everything is correct, it navigates the user to the homepage of its type.

The sing-up component is responsible for the user registration. It is accessible through the login page. This component has a stepper with various forms for each step and user type. The first step is selecting the type of user to be registered. The second step as a form to input all the attributes of the user. The form changes depending on the user type. In case the user is a patient, it has a final optional step, to insert the height, weight and blood type, none need to be filled. Depending of the type of user being registered, it calls a different service registration method and, if the registration was successful, the component navigates to the homepage of the new user. In case the new user is a patient, before it navigates to the new patient’s page, it calls the medical information service to create the patient’s
medical information using the inputted data, if any is submitted, and the medical history service to create its medical history.

The profile component has simple page that presents the user’s attributes and allows for updates.

The components mentioned above are available for all three types of users, the next ones are only available for patients and pertain to the authorizations. The permissions manager component maintains a list of the authorized doctors and organizations of the patient, in this list it allows for updates and removal. There is also a button to add authorizations for new doctors or organizations. Both the add and update buttons open the add doctor permissions dialog or add organization permissions dialog. When the dialog is open via the add button, all fields are empty, while when opened through the update, its fields show the current permissions. The permissions manager component has different methods do add, update and remove permissions for doctors and organizations that calls the same manage permissions method of the users’ service.

Below are screenshots of every component, except the login all are from the perspective of the patient:

![Figure 4.17: User Register Component - Step 1](image1)

![Figure 4.18: User Register Component - Step 2](image2)

![Figure 4.19: User Register Component - Step 3](image3)

![Figure 4.20: User Login Component](image4)

![Figure 4.21: User Profile Component](image5)

![Figure 4.22: Patient Permission Component](image6)
4.10.5.3. Server

The server receives the client requests and calls the proper user methods to handle them. The server’s methods, much like the StateList class receive requests from every user type and use conditionals to choose the correct methods to call and actions to take. These methods use the ID and type of the user making the requests to retrieve its credentials for authentication. After this is done, they send the request to the contract class of the users chaincode. Finally, they send the chaincode’s response to the client.

If the request is to register a new user, the first thing the registerUser method does is call the enroll method where the credentials will be generated, and the user enrolled into the correct certificate authority for their type. After this is successful the register method functions like the ones described above.

4.10.6. Information Access Control

This sub-section starts with an introduction to the class used to implement the access control. Next the way the access control was implemented throughout the multiple chaincodes is described. Finally, the methods of the contract classes and the specifics of their implementation are presented.

The access control implementation used as it’s basis the ClientIdentity of the fabric-shim. The fabric-shim provides an API for lower level implementation of chaincodes. The ClientIdentity class is of great use because it allows access to information pertaining to the client that submitted the transaction. Even if a chaincode is invoked by another, the client this class refers to is always the one who submitted the original transaction to the blockchain. The methods that were used are the getID, getMSPID and getAttributeValue, these return the ID of the client, the ID of their MSP and the requested value of the attribute of the certificate, respectively. The ID of the MSP is used to get the organization to which the client belong to. This is used restrict functionalities and by extension the data to only users of certain organizations. A few examples are: only med labs can create analyses reports; only doctors can update medical information or history; only patients can manage the access control. The ID of the client is used to restrict the access to certain users. For example: only the med lab that created an analyses report can add its results; only the patient can manage access to their data, i.e. the client ID has to match the ID of the user object; a doctor can only update the patient’s medical identity if their client ID is in the authorized doctors list (there is an exception but since it relates to the getAttributeValue it will be explained next). The getAttributeValue is used to get the value of the orgName and specialty attributes. These are custom attributes that every certificate issued by the doctors CA has an relate to the name of the organization (hospital) the doctor works at and their specialty. The orgName serves to check a doctor’s permissions in case they do not appear in the authorized doctors list. The methods will search for the doctor’s organization in the authorized organizations list and, if the organization is found, its permissions will become the doctor’s. The specialty attribute is used to simplify the control for doctors in the
consultation history. It is used to make sure that if a doctor does not have full access to the patient’s consultation history, they can only add a consultation, edit medical data during it or request a specific one, if the specialty matches the consultations specialty.

Since the access control implementation follows a very similar structure throughout the various methods and chaincodes, it will be explained in detail here for all of them. The methods mentioned all belong the contract classes. These were described in the previous sub-sections without reference to access control. The implementation below happens before what was depicted above and in some cases after, always prior to the return of the result. The specifics of each method, and by extension functionality, will be presented next.

The first step on each method is to initialize the client identity object and retrieve the client’s ID and MSP from the newly created object.

Next, a sequence of conditionals is used to filter out users from organizations that cannot use the functionality implemented by that method or access that particular data. If the client is a med lab, unless the methods are on the analyses report chaincode or are user chaincode methods referring to med labs, the transaction would be terminated, and they would receive an error with that information.

After this, the client ID will be used, to check if they have access to the data. If they are patients, the ID would be checked against the medical or patient ID of the object and, if matched the data would be returned to client, if not an error would be thrown, and the transaction ended. If they are med lab, their ID would be compared to the laboratory ID and if they are the same, the data would be returned or saved to the ledger, if not, an error message is thrown to the client, and the transaction will end.

In case the client is a doctor, the access control implementation after the verification of the MSP is more complex. The user object representing the patient would be requested from the user’s chaincode. The list of authorized doctors will be iterated until the doctor is found or the list ends. If an entry of the list has ID equal to the doctor ID retrieved from the ClientIdentity, a new variable called caller will be initialized with the entry and the authorization level for the specific medical data will be checked. In case the doctor does not appear in the authorized doctors list, using the orgName, the same process, will be repeated using the authorized organizations list. In case an ID of on the list matches the orgName, it is this entry that will become the caller and be used to ascertain permission.

If the caller has full permission or permission to only the requested type of data, then the data will be returned. If the transaction is to return a list of objects and the caller does not have full access, each object that could be part of the result will be checked against the permission level for the type and added to the final result list in case the access level allows it. For consultations the attribute is the specialty, for medical history is the status and for analyses reports the type. It is this final filtered list that is returned.

In the instance where neither the doctor nor the organization they work at have any permission or permission for that specific data, an error is thrown to the client.

For doctors the access control only checks the organizations list if they are not found in the doctors list. This allows patients to give doctors a different authorization level than the organizations they work at.

Not all of what is described above is implemented in every method. However, at least some of it is. Next, the specifics of each method will be presented. To avoid repetition, these specifics will focus on how the access control as presented was applied to the specific method, i.e. which parts are present, who has access and what results or actions the permissions allow.

4.10.6.1. Medical Information Contract Class

registerMedicalInformation: using the getMSPID method, it immediately throws an error if the client is not member of the patient organization. If the client is a patient, next it checks if the client’s ID matches the medical ID received as a parameter, if they do not match an error is thrown. If the client is
the patient who owns the medical information, this method will the proceed the way it was described in 4.10.1.1.

**updateMedicalInformation**: with the result of getMSPID, it will end the transaction a throw an error if the client is a patient or a medical laboratory. In case the client is a doctor, the method implements the permission validation for doctors as described above and, if the doctor or the organization they work at do not have permissions for the patient an error is thrown, on the other hand if there is permission the method continues to the actions described in the medical information sub-section.

**removeMedicalInformation**: the access control implementation is exactly like the one of the updateMedicalInformation.

**getMedicalInformation**: since the goal of the medical information was to maintain the most current information on the patient so it could be used not only in consultations, but also in an emergency situation where the patient might not be able to give permission, this method does not implement any form of access control. That being said, since to make the request to the chaincode the server uses the client’s credentials, they have to registered in the platform.

### 4.10.6.2. Analyses Report Contract Class

**createAnalysesReport**: this method uses the getMSPID and getID to ensure that only the med lab who performed the analyses and testing can add the report, by throwing an error and ending the transaction if the client belongs to an organization other than med lab or has an ID different than the laboratory ID parameter value.

**addResults**: the access control implementation in this method works just like the one described above for the creation of an analyses report.

**getAnalysesReport**: if the client is a patient other than the one who owns the report an error is thrown, the same happens for a med lab that did not add the report. For doctors, it checks if the doctor or the organization they work at have any permissions. If a permission is found the authorized analyses report type is compared with the report type and if they match the analyses report object is sent to the client.

**getAnalysesReportByPatient**: if the client is the patient that owns the analyses reports, the list will be sent to the client in its entirety. If the client is a med lab, only the analyses report objects with a laboratory ID that matches the client ID are added to the final result list and sent to the client. For doctors, the permissions lists are check to see if the client or the organization are present in any of them. If so and the doctor as access to all analyses report types, the full list is sent. If there no authorized access to any analyses report type, an empty list is sent back. In case the permissions are only to a specific type, only the reports of that type are added to the final result list that is returned to the client.

**getAnalysesReportByType**: for patients and med labs the implementation is the same as the one described above. For doctors, if they or the organization they work at have an entry on the permissions list, the authorized analyses report type is checked with the requested type. If the types are the same or it as authorization to see reports of all types the list on reports is sent to the client. In other instances, an error is thrown.

### 4.10.6.3. Medical History Contract Class

**createMedicalHistory**: this method implements access control like the registerMedicalInformation, it throws an error if the client is anyone other than the patient who owns the medical history.

**updateMedicalHistory**: the access control in the medical history update method verifies if the client is a doctor and if they or the organization they work for have permissions, if so the update continues as described in 4.10.3.1. If no entry is found on either list or the client is not a doctor the transaction ends and an error is thrown.
getMedicalHistoryByMedicalID: if the client is a med lab the method immediately throws an error. If they are the patient who owns the data the medical history the method proceeds as previously explained. For doctors, if they have any permission, either themselves or the organization the work at, the authorized medical history attribute of that permission will be used to check what part of the medical history can be returned. If the doctor has permission to all the medical history then it is sent back. If on the other hand the authorization is “active” only allergies and medical conditions with active status and medications with end of treatment dates later than the current date are sent in the medical history.

getMedicalHistoryByType: equal to the previous method, med labs do not have any access, patient can only see the medical data of the requested type if they own the medical history and doctors can only see the allergies, medication or medical conditions if they are authorized and only for the status they have permission (all or active only).

getMedicalHistoryByTreatmentDate: for patients they can only receive the list of medications if they own the medical history. The laboratories cannot access any and the doctors receive none if they have no permission or if the end of treatment date is in the past and they only have permission for active data in the medical history, or the full result their authorization level is all.

getMedicalHistoryByTypeAndStatus: for patient and med labs the implementation is the exact same to the one described in getMedicalHistoryByType. For doctors if they or the organization they work at exist in the permissions lists and the authorized medical history attribute matches the requested status they receive the result. In any other circumstance an error is thrown to the client.

getMedicalHistoryByMedicalConditionType: this method implements access control just like the getMedicalHistoryByType.

4.10.6.4. Consultation History Contract Class

createConsultationHistory: if the client is not a member of the doctor organization an error is thrown. If the client is a doctor, the process to ascertain permissions for doctors is followed and if the proper permission exists, i.e. the doctor has permissions to all or that specific specialty, the method continues as describe in the consultation history sub-section.

updateConsultationHistory: the access control is the same as for the create method.

getConsultationHistory: implements access control like the two previous methods.

getConsultationHistoryByPatient: if the client is a med lab or a patient other than the owner of the consultations being requested an error is thrown to the client. If he is the patient who owns the consultations these will be sent. If he is a doctor, the permissions lists will be check to see if either he or the organization they work at have any permission. If so, and the authorization level for consultation history is all, the doctor receives the complete list, if the permission is only for a certain specialty, the list that is returned to the doctor only contains consultations of that specialty. In case the doctor does not have permission to see any consultation an empty list is sent.

getConsultationHistoryBySpecialty: for patients and med lab the access control is implemented in the fashion of what was explained above. For doctors, the process is very similar to the one in consultation history by patient, but after determining if there is an entry in either of the authorization lists the authorized specialty is compared to the requested specialty and if they match or the doctor can access consultation of all specialties the method continues as described in 4.10.4.1. . In any other instance, an error is thrown.

getConsultationHistoryByAttendingPhysician: the only difference between the access control in this method and the other ones of this class, is that in the case that any permission exists for the doctor or the organization, the authorized consultation type is checked with the specialty of the attending physician, and if they match or the doctor has full access, i.e. the authorized type is all, the method continues, if not an error is thrown.
getConsultationHistoryByDate: this method implements access control in the exact same way as getConsultationByPatient.

4.10.6.5. Users Contract Class

registerPatient: using getMSPID and getID the method only continues to the registration in the ledger if the client is a patient and the ID matches the one received as a parameter of the method.

registerDoctor: the same as the one above but for doctors.

registerMedLab: the same as registerPatient and registerDoctor but for members of the med lab organization.

getUser: if the client making the request is the user whose data is being requested then the object is sent. If the client is a doctor requesting a patient’s data, it checks if they or the organization have any permissions for that patient. If so the process continues as described in the users chaincode sub-section.

updatePatient: just like the registerPatient.

updateDoctor: just like registerDoctor.


manageAuthorizations: just like in registerPatient and updatePatient only the patient who owns the data can manage the authorizations.

getUsersByType: given that this method is only called by the manage permissions component of the front-end, it does not implement any access control.

4.10.7. Shared Front-end Components

There are some components, data interfaces and dialogs in the front-end that are shared between the multiple modules.

When the chaincode throws an error, the front-end detects it and opens a dialog with a message to the user indicating the error the occurred. This dialog is shared by all components and the message it displays changes depending on the error and the component that receives the error.

The other shared elements pertain to the allergies, medical conditions and medications. Due to the way chaincodes are packaged and instantiated in the peers, shared classes have to be in each one. Meaning there cannot be a module shared by the different chaincodes. In the front-end however this is possible. With this in mind, the elements that are common to various data types, and the components and dialogs used to allow users to interact with them are shared between them. These are interfaces defining the allergies, medical conditions and medications along with components and dialogs for their creating and display of the details.

An allergy object is defined by the following attributes:

- name: name of the allergy;
- symptoms: description and enumeration of the symptoms cause by the allergy;
- cause: the cause of the allergy;
- date detect: date when the allergy was detected and diagnosed by a doctor;
- status: whether the allergy is still active or has become inactive;
- medications: list with the names of the medications prescribed for the allergy.

A medical condition object has seven attributes:

- type: type of the medical condition, 1 – illness, 2 – injury, 3 – mental illness;
- name: name of the medical condition;
- symptoms: description and enumeration of the symptoms of the medical condition;
- status: indication of whether the medical condition is still active or is now inactive;
- diagnosis date: date when the condition was first diagnosed by a doctor;
• date of cure: date when the medical condition was cured, this attribute should be an empty string as long as the status is active;
• medications: list of the names of the medication prescribed to treat the medical condition.

Medication objects have five attributes, those being:
• name: medication’s name;
• dosage: prescribed dosage of the medication;
• prescription date: date when a doctor prescribed the medication to a patient;
• end of treatment: date when the patient is to stop taking the medication;
• condition: name of the allergy or medical condition the medication was prescribed to treat.

Each of these data types have their own create and a details component. The create components open dialogs and are accessible only by doctors via the create consultation component. They have a form for all the attributes to be field and once that is done and the doctor clicks the add button, the allergy, medical condition or medication is created and sent back to the calling component for it to process the addition to the ledger. The details component opens a dialog with the details of the data it refers to. For patients, it only displays the details for patients but allows update for doctors. For the update it has methods that directly interact with the medical history service to send the updates to the ledger.

Below are screenshots of the create and details dialogs for the allergies, medical conditions and medications from the perspective of the doctor:
4.11. Documents

While in the original plan there was a reference to supporting documentation that could take the form of an instructions manual on how to install and use the platform, it was decided at a later date that such a document would not be needed. Considering this the documentation phase consisted in documenting the code and production of a presentation of the platform for the host organization.

All code produced for the chaincodes, server and front-end web app are properly and fully documented.

The presentation includes an explanation of the users, the functionalities and roles they have in the platform as well as the types of medical data stored in the blockchain. Besides this, a demonstrative video and the timeline of work are also included.

4.12. Tests

During the development of the platform tests were conducted to ensure that the implemented functionalities worked as expected and fulfilled the defined requirements. These tests consisted of simulating use of the platform by inputting data and making requests and comparing the returns and output of the console logs on the chaincode containers with what was expected. When those differed from what was anticipated an error was detected and would be debugged. After this debugging more tests would be conducted to ensure the problem was fixed and no new ones were created by the correction. These tests had as a goal not only testing if the data was correctly added, retrieved from the ledger and sent to the client, but also to guarantee that the access control, updates in-between chaincodes and other actions and validations functioned as they were designed.

Due to the context in which the platform was developed and resources available, performance and load tests were not conducted. The entire platform, chaincode, web-app and server, ran locally on a virtual machine installed on a Windows laptop. This is important because blockchain solutions, including Hyperledger are designed to run in multiple different machines and not in a single one. With this in mind it is logical that the performance of the platform in the current setup would never be representative of the real performance of a real-world implementation and as such the results of performance tests would be inaccurate. The same rational justifies the reason has to why load testing was forgone. The inaccuracy of these tests would not be an overestimation of the true abilities of Hyperledger Fabric for this type of system, but a considerable underestimation, that would lead to a negative evaluation of the platform.

While the performance requirements could not be tested, the usability, functionality are reliability could. Besides the tests conducted during development, after the platform was completed it was tested by two people not involved in its creation. Each of the testers performed the same four tests, designed
to cover all the functionality requirements across the various types of users and user functions. For the
tests, the med lab and doctor were already registered in the platform. These tests were:

- **Test 1:**
  1. Register a new patient;
  2. Give doctor Drake Ramore permissions to all the medical data;
  3. Give Beth Israel Hospital permission for only general medicine, no analyses report types and only active parts of the medical history.

- **Test 2:**
  1. Login to Stark Medical Labs, ID: 162534 and password: 1423;
  2. Create an analyses report for the patient created in test 1 with type blood test.

- **Test 3:**
  1. Login as doctor Drake Ramore, ID: 654321 and password: 4321;
  2. Start a consultation for the patient created in test 1;
  3. Search for analyses reports of type blood test and open the results file;
  4. Update the patient’s blood pressure;
  5. Register the consultation to the ledger.

- **Test 4:**
  1. Login as the patient created in test 1;
  2. Request the complete consultation history and open the consultation registered in test 3.

By observation of the testers process in accomplishing each action and by the opinions given by them at the end, some conclusions were taken. The only issues mentioned by the testers were that the platform’s interfaces were in some circumstances not intuitive and that sometimes it did not provide the user with enough feedback on the success or failure of an action. As for the functionality and reliability requirements, no issue was raised, the data was always available, accurate and all the functions tested for the multiple users worked as expected without problems.

From the observations it can be concluded that the platform meets the functionality and reliability requirements stipulated. As for usability, the issues raised did not impede use.

More in-depth analyses of how these tests impact the platform and what can be concluded about the applicability of blockchain to a healthcare solution is presented in the section below.

### 4.13. Platform Evaluation

The objective of the platform was to test the applicability of blockchain, in particular Hyperledger Fabric, to solve the problems in the healthcare industry mentioned in previous chapters of the report. In a general overview, these problems relate to the heavy decentralization of information in the industry, the lack of communication and of trust between the different medical organizations and their members. The second point causes the burden of transmitting the relevant information to the person seeking medical attention, who in some circumstances might be unable to transmit it. The lack of trust leads to the request of redundant tests and analyses, which cost money and wastes time in an industry where diagnoses are time-sensitive. Another problem this platform was designed to address is the location where the medical data is stored. While lack of trust in the safety and privacy of one’s data is not exclusive to medical data, it is still of great concern to patients.

The platform was then designed and implemented in order to best address these issues. By nature of the underlying technology, there are a whole host of issues that are immediately resolved. The distributed ledger ensures that every member of the network has an up-to-date copy of the data. The consensus mechanisms guarantee that no malicious actor introduces data into the ledger, thus establishing
trust in the data. There is also an assurance that once the data is inserted it cannot be altered, meaning that the medical data is always accurate. Finally, considering that a copy of the ledger is stored by each of the nodes in the network, it eliminates the need for third party data storage.

The chosen implementation of blockchain, Hyperledger Fabric also provides some extra features that can address the remaining problems. Fabric allows for the network to be setup with different configurations of not only its architecture, i.e. peers, orderers, channels, etc. but also in terms of the consensus mechanism and the way blocks are created. The consensus mechanism or the manner in which blocks are created was not changed, because the network topology was very simple, and all members were controlled by the developer. The most important feature of Fabric for this platform is the ability to implement business logic using smart contracts, chaincode. Through smart contracts, rules regarding the treatment of the data from creation to access were implemented. While the contract methods and functionalities regarding the creation and data requests are very important for the platform to have any practical use, these are not much different from what can be implemented for a traditional relational database. The principal difference is the way the storage and access to the data is made and sent between the back-end and the server handling user requests. Where smart contracts really help in solving the problems mentioned above is in the access control. Using the ClientIdentity class to retrieve information about the client from their trusted certificate, it is possible to allow not only access to the data to certain users but also the functionalities available, all of which changes depending on the owner of the data and what permissions they give. By default, no user other than the patient who owns the data can access or add to it. Only by specifically giving a doctor or medical organization permission can they access the medical data of that patient. This gives patients a great degree of trust because they know exactly who is authorized to add, update and search their entire medical data, and they can also change or completely remove those authorizations. Due to the way the access control is implemented, patients can give the doctors they know and trust far greater access then the organizations in which they work.

To make sure the platform can take advantage of what the technology and the selected implementation have to offer to solve the problems presented, a set of requirements and constraints were identified. The functionality requirements where all the functionalities were presented are all implemented in the platform, both for front and back-ends. The usability requirements are not as relevant for the application of blockchain to the healthcare industry, because they refer to the user interface and that is not a factor for the validity of the technology that is exclusively back-end. That fact notwithstanding, the vast majority of the requirements are met and some partially. These requirements were primarily tested in the user tests discussed in the previous section, as such they will be discussed in greater detail in a future paragraph when those tests’ results are explained. All of the reliability requirements are achieved by the platform, due the very nature of blockchain. The performance requirement identified was not verified, in the tests performed. This does not mean however that it cannot be accomplished, as explained in the tests section performance testing was not done due to the inaccurate results that would be obtained because of the environment in which the platform was running. As such, it is entirely possible that in a more suitable environment that requirement is not only achieved but superseded. Regarding the supportability requirements, the platform was designed in a modular way that allows ease of extension and implementation of new features. As for its testability, due to inexperience in blockchain development, the available resources and development context, the only tests conducted were supplying multiple inputs and tests with users that mostly looked at usability and reliability. This does not mean that the platform failed to be testable as was described in S1, instead it means that it is not possible to conclude. The constraints are related to the technology, programing languages and frameworks to be used and these were all met.

The final thing that will be looked at to evaluate the platform is the tests conducted, more specifically the tests with users. While the other tests were important to identify and fix errors and software bugs, they were also conducted by the developer which means they were not totally impartial. The four
tests with users were presented above along with the observations and a brief conclusion. These tests looked at functionality, usability and reliability. In terms of functionality, they had users perform multiple actions that covered all six requirements, with all being functionalities being easily used by the testers. For reliability, the data inputted by the users was accurately stored and when requested was available and with the same values as when initially introduced to the ledger. Recoverability was not tested because no node of the network crashed during any of the tests. The usability of the platform was, by observation and comments made by the testers, good but with room for improvement. The main issue was the lack of feedback, regarding success of a transaction or while waiting for a transaction to finish. Another point that was mentioned was that finding a function in the more complex interfaces is not always intuitive. None of these however detracted from the use of the platform and have no bearing on the applicability of blockchain. Some problems with the usability requirements were expected, because I have very little experience in front-end development. Since as developer I was the one developing the front-end, it was designed more to allow testing and use of the platform and not with commercial use in mind.

Considering all the points above, it is possible to conclude that the platform met its goals. While on a very small scale, it implements a healthcare system that uses blockchain to store the medical data. Through the business logic implemented in the chaincodes and the features of Hyperledger Fabric the problems identified are addressed.

Despite the positive results, there is still a lot that would need to be done to make this a commercially viable platform. In the next chapter there is a discussion on potential applications of a system that uses the developed platform as a proof-of-concept and future work needed to improve the platform and make more secure and take advantage of more of Fabric’s features.
Chapter 5. Conclusion

This chapter contains a short summary of the work developed, a discussion of the results of the platform evaluation and the potential of blockchain for the healthcare industry and possible future work to further improve the value of the developed system.

5.1. Work Summary

The work started with a study of blockchain and an analysis of different industries with problems that could, potentially, benefit from the technology. This knowledge was then used to identify use cases in which blockchain would truly be helpful and not just an experiment with no real-world utility. Using a set of criteria provided by the host organization, the use cases were evaluated. It was based on this ranking that the healthcare use case was selected. The scope of the use case was narrowed to ensure that a proof-of-concept could be implemented in the timeframe and context of the project.

After clearly defining the use case, the next phase was selecting the blockchain implementation that would be used for the platform to be developed. For this, a number of possibilities were studied and compared with one another. In the end, Hyperledger Fabric was chosen as the most adequate.

Before the implementation started, the platform’s architecture, requirements, data and user types, along with the functionalities were defined. Then, the topology of the blockchain network was created, this was done following the documentation and tutorials available in the Fabric website. This topology was adapted to make sure that not only would the platform follow the architecture and criteria, but also in a way that was not overly and unnecessarily complicated for the scale of the use case.

With the platform well defined, the next stage was its implementation. Following the Fabric tutorials, the network was created. Once this was done, the development of the chaincodes, server and front-end began. This was the longest stage of the entire project. The requirements and architecture were carefully followed during implementation. Though some changes to these were made when it became apparent that they were required for the platform to work as planned.

Both during development and after, tests were conducted. These tests had the goal of finding errors and bugs in the platform, as well as, verifying if the requirements were met by the system.

The results of the tests were used in conjunction with observations from the development and comments from multiple meetings with the Accenture team to evaluate the platform and with that evaluation discuss the potential of blockchain for the healthcare industry.

5.2. Discussion

As indicated in the platform evaluation section, the final platform achieved the goals it was designed to. While this is a very good sign, the scope of the problems and functionalities is very small compared to the original use case. Moreover, the development and testing environment were isolated from the real-world context in which a system such as this would operate. All of these factors mean that conclusions on the greater applicability of blockchain for the healthcare industry are not direct conclusions based on the results but based on logical analyses of the results of the platform and knowledge of the technology’s potential.

The platform implementation phase took around four months. In that time, it was possible for one developer, with no prior experience, to implement a significant number of functionalities in the chaincode while also developing the server and front-end. It stands to reason that a team of both front-end and experienced blockchain developers could implement a far more sophisticated platform. This
system, that would use the platform as a proof-of-concept and basis, could have a more complex blockchain network topology and better developed chaincode to more efficiently answer the problems identified in this report.

Something that was not explored or changed during development were the configurations of the network, such as the block formation or the consensus mechanism. This part of the blockchain network configuration, in particular using Hyperledger Fabric, can be very important in a real-world solution because it can be used to set up rules agreed upon by the different participant organizations.

As explained before, load and performance testing were not done because of the environment in which the platform was running. A real-world implementation would have platform’s three constituent parts, the blockchain, the server and the web-app running in different machines. This would, depending on those machines and the connection speeds, create a system with performance levels at least comparable to current systems that use relational databases. Other important benchmarks, like transactions per-second are limited by the technology, but these tend to increase with each new release of Hyperledger Fabric.

Considering all of these factors, and the inherent advantages of blockchain referenced throughout this report, it is possible to conclude that there is promise in blockchain to solve the problems identified in the healthcare industry.

The original use case describes a scenario in which both private and public healthcare providers participate in the network and share medical data of the patient’s, accessing it only when given permission. This is a vision of the future that could potentially take decades to implement. However, with the knowledge gained another more immediate application was identified. Let’s consider a private healthcare provider that runs a network of different hospitals and has partnerships with several medical labs. This provider could setup a solution based on the platform developed for this dissertation. The doctors employed would be registered into the platform, along with the partner labs and patients. The patients would control access to the doctors and the multiple hospitals in the network by, for example, giving higher permissions to the doctors they know and more basic permissions to the hospital(s) they go to while completely blocking access to unfamiliar doctors and hospitals in the provider’s network. If the patient has a need to go to a new doctor o and/or new hospital, appropriate permissions could be given during that specific consultation. This ensures that the necessary medical data is always available during consultations but also guarantees to the patients that doctors employed by the provider cannot access data in the network with prior approval.

5.3. Future Work

Despite achieving its goals and serving as a proof-of-concept for a blockchain system in the healthcare industry, there are still several improvements that could be made to the platform. Most of these were not added due to time constraints or insufficient knowledge for their implementation. In this section of the report these improvements will be presented.

Thanks to the modularity and extensibility requirements it is fairly easy to implement new functionalities in the platform. The addition of a vaccination record, a history of emergency room visits, similar to the consultation history, are just a few of the possible data types that could be added to platform. By increasing the types of medical data in the platform, it would provide a better picture of a patient’s history and allow for more accurate diagnoses. Besides adding new data types, existing ones can be expanded to include more information.

Another area in which the platform can certainly be improved is in the topology and configuration of the network. If it is to be implemented and used in a real-world situation, configurations such as the consensus mechanisms or block formation need to be changed from the basic default settings. These new configurations can be designed with the assistance of the stakeholders, in particular the clients, to
reflect the rules and relationships that these will have with the data and with each other. The network’s topology, i.e. the channels, peers and organizations, etc. can also be improved for the same reason.

During implementation instead of the standard getState and putState methods for storing and retrieving data from the ledger, the methods getPrivateData and putPrivateData were considered. These latter two methods use collections to define access to the private data. This private data is stored in a separate private ledger in each authorized peer. Using these collections, the medical data could have a public and private format with the collections defining who can access the private data. An attempt was made to implement these methods in the StateList classes but an error with the collection file prevented the chaincode from being installed and instantiated. Due to this and considering that access to the data was already going to be restricted with the access control, the private data methods were abandoned for the getState and putState. However, if work was to continue the change to private data is definitely something that should be explored and implemented in order to add an extra layer of security to the medical data.

Improvements or a complete overhaul of the web-app would also need be considered. As mentioned before, front-end development and design are not my area of expertise and as such the web-app is very basic and generic. There were also some of the comments and observations during the tests with users that noted that some changes were needed to improve the usability. For a wider and real-world use, a change in the web-app would without question be required.

In line with the previous point both the server and the chaincode could also benefit from some improvements. Expertise in chaincode development was non-existent at the beginning of the project, as such some decisions in the implementation that were made are not the optimal or the most efficient. A review of the code with more informed knowledge without a doubt will identify some sections that can be improved. The same happens for the server, while there was some knowledge of the server development, this was nowhere near as extensive as needed to implement a fully functional server for a fully-fledged platform. The implemented server is based on the one provided with the tutorial code. It works well and performs all the necessary tasks, but a review and improvement of it could only increase the value of the platform.

While all the points above are significant, the first and definitely most important thing that should be done in the future is moving the platform form the current environment and into a more adequate and realistic setup. By doing this, performance and load tests could be done and with the results a true picture of what the technology is capable of would become apparent.
**Bibliography**


