INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




Sistema de informação laboratorial para o COVID-19

BRUNO DANIEL ALVES ROCHA
Outubro de 2021

POLITÉCNICO
DO PORTO

Instituto Superior de Engenharia do Porto

Laboratory Information System for COVID-19

Bruno Daniel Alves Rocha

A dissertation submitted in partial fulfillment of the requirements for the degree of Master of Science, Specialisation Area of Software Engineering

Supervisor: Dr. António Constantino Lopes Martins
Co-Supervisor: Dr. Luís Miguel Pinho

Evaluation Committee:
President:
Dr. , Professor, DEI/ISEP

Members:
Dr. , Professor, DEI/ISEP
Dr. , Professor, DEI/ISEP
Dr. , Professor, DEI/ISEP

Porto, October 15, 2021

Dedicatory

To my parents, sister and girlfriend who are always there for me

To my family, friends and everyone that believes in me

Special thank you to my grandfather

Abstract

COVID-19, a respiratory disease caused by SARS-CoV-2, first appeared in Wuhan, China, on 31 December 2019. It has since spread worldwide and developed into an ongoing pandemic. Currently, COVID-19 does not have a cure, and prevention is the only way to fight against it.

During waves of higher infection cases, tracking the infected population becomes a difficult but crucial task. Only a COVID-19 test can diagnose a person, and RT-PCR tests are the most effective.

PORTIC, the research centre for P.Porto, started using its laboratory for RT-PCR tests to diagnose COVID-19 for the P.Porto community and some health centres that belong to ARSN. During this process, the laboratory needs to manage all of the sample and testing information and report the test results. This information management became a burden, and the staff would lose most of the time with administrative tasks.

This dissertation's main objective is to develop a laboratory information system for PORTIC. This system must satisfy the elicited and specified requirements. For that purpose, multiple architectures were analysed, concluding that the clean architecture is the best option for this system.

The system supports data importation from multiple external sources, report generation and exportation and the entire sample flow. Its development followed a scrum methodology where each requirement was validated through user acceptance tests at the end of each iteration.

To evaluate the system's success, the laboratory answered a questionnaire to determine the perceived usefulness and ease of use. This concluded that the system was successful since the questionnaire determined that it was extremely useful and easy to use. The developed system is an innovation on COVID-19 testing since there are no real options in the market, and different laboratories can reuse the system to tackle COVID-19 testing.

Keywords: COVID-19, healthcare, laboratory, LIS, LIMS, information system, software architecture, Clean Architecture

Resumo

A COVID-19, uma doença respiratória causada pelo SARS-CoV-2, apareceu pela primeira vez em Wuhan, China no dia 31 de dezembro de 2019. Desde então, esta doença espalhou-se por todo o mundo, devenvolvendo-se numa pandemia em curso. Atualmente, não existe cura para a COVID-19, sendo que a única maneira de resistir à doença é através da prevenção.

Durante as ondas de grandes números de infeções, rastrear a população infetada transforma-se numa tarefa árdua mas fundamental. A única maneira de diagnósticar a doença é através de um teste de COVID-19, sendo que os testes de RT-PCR são os mais eficazes.

O PORTIC, centro de pesquisa do P.Porto, começou a realizar testes de RT-PCR, no seu laboratório, para diagnosticar COVID-19 à comunidade do P.Porto e para alguns centros de saúde que pertencem à ARSN. Durante este processo, o laboratório precisa de gerir toda a informação sobre as amostras e os testes, assim como reportar os resultados dos testes. Esta gestão de informação tornou-se num incómodo e os funcionários passaram a perder a maior parte do seu tempo com tarefas administrativas.

O objetivo principal desta dissertação é o desenvolvimento de um sistema de informação de laboratório para o PORTIC. Este sistema deverá cumprir os requisitos elicitados e especificados. Para esse propósito, foram analisadas diferentes arquiteturas, chegando-se à conclusão de que a clean architecture é a opção mais viável para este sistema.

O sistema suporta importação de dados de múltiplas fontes externas, geração e exportação de relatórios e todo o fluxo de amostras. O desenvolvimento do sistema seguiu uma metodologia scrum onde cada requisito foi validado através de testes de aceitação do utilizador no final de cada iteração.

Para avaliar o sucesso do sistema, o laboratório respondeu a um questionário para determinar a utilidade e facilidade de utilização percecionada. Isto concluiu que o sistema foi bem sucedido dado que o questionário determinou que foi extremamente útil e fácil de utilizar. O sistema desenvolvido é uma inovação em testes de COVID-19 pois não existem opções no mercado e outros laboratórios podem reutilizar o sistema para endereçar os testes de COVID-19.

Palavras-chave: COVID-19, cuidados de saúde, laboratório, LIS, LIMS, sistema de informação, arquitetura de software, Clean Architecture

Acknowledgement

I want to thank everyone that contributed to this dissertation; you made it possible.

To Professor Pilar Baylina and Professor Rúben Fernandes, from the PORTIC laboratory, for their invaluable help during the project.

To Dr. António Constantino Martins and Dr. Luís Miguel Pinho for their supervisory and guidance with the project and this dissertation.

Finally, a big thank you to my family, my girlfriend and my friends, without them I would not be able to do this.

Contents

List of Figures

List of Tables

List of Acronyms

ARSN	Administração Regional De Saúde Do Norte.
CQRS	Command Query Responsibility Segregation.
CQS	Command Query Separation.
FFE	Fuzzy Front End.
HIS	Healthcare Information System.
IMIA	International Medical Informatics Association.
IT	Information Technology.
LIMS	Laboratory Information Management System.
LIS	Laboratory Information System.
MVC	Model-View-Controller.
NCD	New Concept Development.
P.Porto	Politécnico do Porto.
PORTIC	Porto Research, Technology & Innovation Center.
QFD	Quality Function Deployment.
RT-PCR	Reverse transcriptase-polymerase chain reaction.
SARS	severe acute respiratory syndrome.
SARS-CoV-2	severe acute respiratory syndrome coronavirus 2.
SUT	System Under Test.
UAT	User acceptance test.
UI	User Interface.
WHO	World Health Organization.

Chapter 1

Introduction

This chapter describes the context of the dissertation, the problem presented, the objectives to complete and the expected results, as well as the structure of this document.

1.1 Context

On 31 December 2019, in Wuhan, China, the new coronavirus disease, named COVID-19, was identified. It is a respiratory disease caused by severe acute respiratory syndrome coronavirus 2 (SARS-CoV-2), and it has, since, spread worldwide, developing into an ongoing pandemic.

The most common symptoms are fever, dry cough and fatigue (World Health Organization 2020). However, at least one-third of the infected population is asymptomatic, does not develop any symptoms (Wang et al. 2020). Among those who develop symptoms, 81% have mild symptoms and recover without hospital treatment. 14% have severe symptoms and require oxygen. And 5% are classified as critical, needing intensive care (Wu and McGoogan 2020) (World Health Organization 2020). The virus spreads mainly through respiratory droplets when an infected person coughs, sneezes or even talks or sings. It can also spread via contaminated surfaces.

COVID-19 does not have a cure, and the only way to fight it is through prevention. The Centers for Disease Control and Prevention (CDC) (2021) has a set of guidelines for the prevention, and those include wearing a mask, maintaining social distancing, avoiding crowds, washing hands frequently, among others. The most recent prevention method is the vaccine. According to the CDC (2021a), there are two vaccines authorised and recommended to prevent COVID-19, the Pfizer-BioNTech COVID-19 vaccine and the Moderna COVID-19 vaccine. "The Pfizer-BionNTech COVID-19 vaccine was 95% effective (...) in preventing symptomatic laboratory-confirmed COVID-19 in persons without evidence of previous SARS-CoV-2 infection" (Oliver et al. 2020), while the Moderna COVID-19 vaccine was 94.1% (Oliver et al. 2021).

Reverse transcriptase-polymerase chain reaction (RT-PCR) is the test used to diagnose a person with COVID-19 (WHO 2020). Although this test effectively identifies positive cases, it is less accurate on negative ones, leading to multiple false-negatives (Zitek 2020).

Another test used for the COVID-19 diagnose is the antigen test. It is less expensive than the RT-PCR and returns results in approximately 15 minutes. This test's best usage is on a high incidence of infection in the community, where it can detect COVID-19 faster. However, the antigen test is less accurate than the RT-PCR, and an RT-PCR might be necessary to confirm COVID-19 (CDC 2021b).

1.2 Problem

Porto Research, Technology & Innovation Center (PORTIC), the research centre for Politécnico do Porto (P.Porto), started doing RT-PCR COVID-19 tests for the P.Porto community and a few health centres that belong to Administração Regional De Saúde Do Norte (ARSN). Only samples from the P.Porto community are collected, since health centres collect their own, sending them to the PORTIC laboratory. PORTIC is responsible for determining the test result, reporting them to ARSN for the health centres samples. As for the P.Porto community samples, PORTIC directly notifies the person tested about the outcome.

The P.Porto community applies for the COVID-19 test through an online form where they insert the required information. Latter, the person is informed of the date for the sample collection. Upon the sample collection, the laboratory identifies the sample with a COVID code and a laboratory code. The COVID code represents the sample at a national level and is mandatory. While the laboratory code exists because it is smaller than the COVID code, fitting the sample and making it easier to manage for the laboratory. The sample goes to the laboratory and is analysed to determine the test result, taking up to 48 hours.

The health centres send a list of the samples to be analysed, which already contains a COVID code. The laboratory receives the collected samples, where they assign a laboratory code and analyse it.

During the process of analysis, the laboratory only knows the laboratory code of the sample. After the test result, they have to map the laboratory code back to the COVID code and produce the report to ARSN or notify the person from the P.Porto community.

This process became difficult to manage, especially during waves of higher COVID-19 infections in the community. One significant problem is the constant mapping of the laboratory code to the COVID code and vice-versa. The laboratory manages all of the data through multiple files, affecting the data consistency. Another problem for the laboratory is the production of reports for ARSN and to notify the P.Porto community.

1.3 Objectives

This dissertation's main objective is to build a system that allows the PORTIC laboratory to manage all the information of the COVID-19 tests.

The system must support data importation from different external sources. The data arrives from various health centres and the P.Porto community online forms, in different files, and needs to be normalised.

The system must support the importation of the results that the RT-PCR supplies. It must be able to identify the samples by COVID code or laboratory code. And it must support report generation about the COVID-19 test results.

The objective described will be achieved through the following tasks:

- **Requirements elicitation and specification:** application of elicitation techniques and requirement specification with the stakeholders;
- **Investigation and analysis of existing solutions to similar problems:** market research and comparison of existing solutions for similar problems;

- **Investigation and analysis of different architectural designs:** analysis and comparison of multiple alternatives to the architectural design of the solution;
- **Identify and justify the most viable alternative:** determine the most viable alternative according to the functional and non-functional requirements specified;
- Develop a solution that supports:
  - **Data importation from different external sources:** users must be able to import data from multiple health centres and from the P.Porto community form;
  - **Importation of RT-PCR test results:** users must be able to import the test results from the RT-PCR;
  - **Report generation:** users must be able to generate reports for ARSN and P.Porto community;
  - **Authentication and authorisation:** the solution must support different roles in the system;
  - **Management of users authorisations:** the solution must support authorisation management to users of a specific role.
- Implement tests:
  - **Unit:** validates the behaviour of the smallest units of functionality;
  - **Integration:** validates the behaviour of multiple combinations of code units and how they work together;
  - **Acceptance:** validates the behaviour of the entire system and guarantees that the requirements are satisfied.

1.4 Expected Results

By the end of this dissertation, the following results should be achieved:

- **Data importation from different external sources:** users should be able to import data from multiple health centres and from the P.Porto community form;
- **Importation of RT-PCR test results:** users should be able to import the test results from the RT-PCR;
- **Report generation:** users should be able to generate reports for ARSN and P.Porto community;
- **Users management:** specific users should be able to manage users authorisation;
- **Tests implementation and realisation:** the system should have automated tests to guarantee the system's correct functionality.

1.5 Document Organisation

This document is organised into the following chapters:

- **Introduction:** presents the context, problem, objectives and expected results of this dissertation, as well as the document organisation;

- **Context and State of the Art:** describes in detail the context of this dissertation, analyses the state of the art and compares market solutions for the current problem;
- **Value Analysis:** analyses the value of the current dissertation, applying the New Concept Development (NCD) model, the value proposition canvas and the Quality Function Deployment (QFD);
- **Analysis and Design:** details the requirements for the system to develop, relevant technologies for its development and the architectural design of the solution.
- **Implementation:** explains the code structure, analyses the design patterns used, describes functional and non-functional requirements and demonstrates the tests done;
- **Evaluation:** evaluates the system's perceived usefulness and ease of use and compares the system with the ideal Laboratory Information System (LIS);
- **Conclusions:** resumes this dissertation and presents the objectives accomplished, along with its limitations, future work and a final assessment.

Chapter 2

Context and State of the Art

This chapter presents the current state of the art in medical informatics, information systems, medical information systems and Healthcare Information Systems (HISs). It also analyses multiple market solutions for the current problem and how they compare with each other.

For the research, the author used Google Scholar, PubMed and ResearchGate as research engines. He selected multiple keywords as "information systems", "healthcare", "medical", "laboratory" and "COVID-19" and made search terms with each keyword or even combinations of them.

For the first search, the author read the summary, keywords and conclusion/results of the papers found. This fast-reading allowed the author to narrow the number of articles. Next, the author read the article to understand its contributions, taking notes on them. If the article was a literature review, the author would research for its references. This process was reiterated with new keywords, new combinations of search terms and more refined search terms.

2.1 Medical Informatics

Medical informatics or healthcare informatics is the study, development and implementation of computer applications to improve medical care (Sami 2019). Bemmel (1984) defines it as a comprise of "the theoretical and practical aspects of information processing and communication, based on knowledge and experience derived from processes in medicine and health care".

2.1.1 History

In 1959, Ledley and Lusted (1959) published an article analysing reasoning processes inherent in medical diagnosis. They explore the idea of computer usage as an aid to medical diagnostic. The idea is not to use the computer to determine the value of the treatments involved, as they must be evaluated and judged by a physician. But to use it for decision-support, enabling a more precise diagnosis.

In 1985, Willems, Arnaud, et al. (1985) established a reference library for evaluating computer electrocardiogram measurement programs. This paper significantly contributed to the standardisation of computer-derived electrocardiogram measurements. In 1991, Willems, Abreu-Lima, et al. (1991) published a systematic assessment of multiple computer programs for the interpretation of electrocardiograms. They concluded the median percentage of correct classifications was 6.6 per cent lower for the computer programs than for the

cardiologists. However, some programs "performed almost as well as the cardiologists in identifying seven major cardiac disorders".

In 2000, Arokiasamy et al. (2000) published, on behalf of International Medical Informatics Association (IMIA), the first international recommendations on education in health and medical informatics. They point out a gap in the knowledge of healthcare professionals about medical informatics. And only improved education of healthcare professionals can fill this gap. The paper also clarifies the importance of health and medical informatics education, claiming that:

- progress in information processing and technology is changing society;
- the amount of health and medical knowledge is increasing at a fast rate, and only information technologies can keep up with it;
- there are significant economic benefits from the use of information technologies;
- medical informatics enhance the quality of healthcare;
- the developments of medical informatics will at least continue at the same pace as could be observed;
- healthcare professionals that are well-educated in medical informatics are needed to process information;
- provision of high-quality education on medical informatics will raise the quality and efficiency of healthcare.

This publication "is a clear sign of the international presence and maturity of medical informatics as a discipline" (Haux 2010).

Medical informatics became a common topic and is utilised in multiple ways. According to Prokosch and Ganslandt (2009), most university hospitals have implemented hospital information systems. These systems were collecting big amounts of medical records. He identified an opportunity to explore this data for research purposes, making use of data warehousing and data mining techniques.

2.1.2 Information Systems

An information system is a system capable of collecting, processing and storing data for providing information to the users (Zwass 2020). It splits into five components:

- **hardware**: the physical components of the system;
- **software**: the application or set of instructions that run on and tell the hardware what to do;
- **data**: a collection of facts used by the system to produce information;
- **people**: an essential component often overlooked. Includes users, administrators, programmers and every person that interacts with the system;
- **process**: a series of steps necessary to achieve an outcome (D. T. Bourgeois, Ph.D., and Bourgeois 2014).

Information systems enhance organisational capabilities. They support business operations, decision making and relationships with customers, suppliers and partners. These systems

can lower the costs of communication inside an organisation and improve supply chain coordination (Zwass 2020). The implementation of an information system, however, is not a guarantee of competitive advantage. Carr (2007) says that, for a brief period, information technology "opened opportunities for forward-looking companies to gain real advantages. But, as their availability increased and their cost decreased - as they became ubiquitous - they became commodity inputs".

The use of information systems enables new organisational structures. Virtual organisations where employees can work from anywhere, and not just the office, are now possible (Zwass 2020). During the ongoing pandemic, these organisational structures have been proving to be valuable. Organisations that fit into these structures quickly adapted to the new reality of working at home without a significant impact on the profit. Some organisations even managed to increase the profit and value of the company. While other organisations, who were not ready for this structural change, got at risk of losing the entire business.

2.2 Health Information Systems

"Information systems have great potential to reduce healthcare costs and improve outcomes" (Fichman, Kohli, and Krishnan 2011). These systems can improve the coordination of healthcare and support decision-making. They can be a great tool when dealing with outbreaks of infectious diseases, such as severe acute respiratory syndrome (SARS) or SARS-CoV-2. These diseases threaten many lives, and coordinating health resources becomes a race against time since controlling the spread is as important as treating it (Fichman, Kohli, and Krishnan 2011).

Fichman, Kohli, and Krishnan (2011) researched the role of information systems in healthcare. They explain that healthcare information is highly personal, and the digitisation of health information has several benefits. HIS enable the providers to share electronic health records leading to higher administrative efficiency, lower healthcare costs by eliminating unnecessary medical tests, and fewer medical errors.

They also point out some barriers to healthcare technology adoption. One of them is that powerful actors in healthcare often resist technology since they associate other activities besides patient treatment as administrative inconveniences. Another barrier detected is the multidisciplinary nature of healthcare. Different professionals involved in healthcare have different perceptions and usage of technology, which adds another layer of complexity to the implementation of a HIS.

2.2.1 Failure and success

A HIS that succeeds in one setting may fail in another. Learning and adapting are the keys to a new information system. They are necessary to determine the best adaptation of the technology and the organisation to achieve a good fit. Ultimately, these adaptations must incorporate into the organisational routines and ensure continuous improvement (Fichman, Kohli, and Krishnan 2011).

Dissatisfied with the analysis of HIS failures, Heeks (2006) developed a new model to understand its failures and, therefore, its successes. He categorises the HIS failures in three:

- **the total failure**: an initiative that was never implemented or immediately abandoned;

- **the partial failure**: an initiative where major goals are unattained or where there are undesirable outcomes;
- **the success**: an initiative that attains most stakeholders goals, and there are no undesirable outcomes.

He correlates the failure of a HIS with a gap between the design and the reality. That is, "the amount of change between "where we are now" and "where the HIS wants to get us"". He refers to this as the design-reality gap.

2.2.2 The design-reality gap

The design-reality gap has two stakeholders: the designers who design the HIS and the users who populate the local reality. It consists of seven dimensions, which are entitled ITPOSMO: Information, Technology, Processes, Objectives and values, Staffing and skills, Management systems and structures and Other resources (Heeks, Mundy, and Salazar 1999). Figure 2.1 illustrates these dimensions.

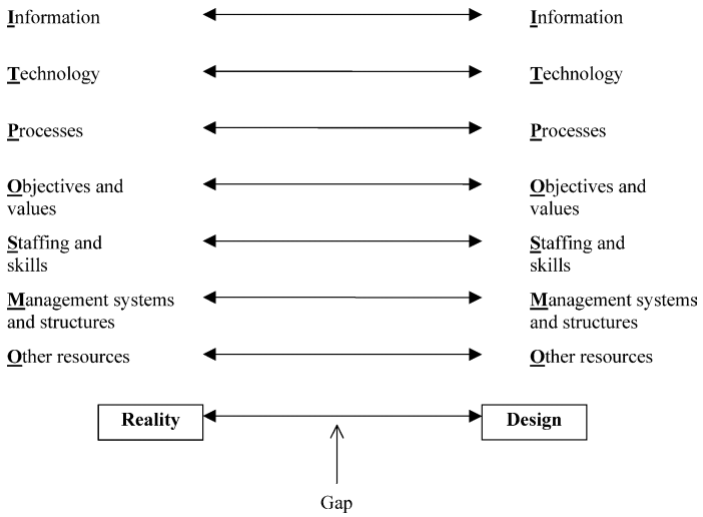


Figure 2.1: The ITPOSMO dimensions of the design-reality gap model (Heeks, Mundy, and Salazar 1999)

Large gaps increase the likelihood of HIS failure. Heeks (2006) highlights some archetypes of those large gaps: the hard-soft gap; and the design-implementation context gap.

A hard-soft gap happens when there is a collision between hard rationalities[1]. It also happens when hard rationalities conflict with the softer realities of the user group. While a hard-soft gap happens on an intra-organisational level, a design-implementation context gap happens on an inter-organisational one. Heeks (2006) gives examples like the public-private sector gap, where the public and private sectors have significant differences. An information system designed for one of them is highly likely to fail if introduced to the other. Another example is the country gap, where developing countries and industrialised countries also have significant differences.

[1] A hard rationality is the rationality of an entity that plays a central role in a HIS design (Heeks 2006).

To summarise, the design-reality gap model is useful as an evaluation tool, determining the failure or success of a HIS and as a risk management tool, identifying major sources of risks in a HIS project.

2.3 Laboratory Information Systems

A Laboratory Information System (LIS), also referred to as Laboratory Information Management System (LIMS), is an information system with features to support laboratory operations. According to Sepulveda and D. S. Young (2013), LISs are essential to manage the flow of information between health care providers, patients and laboratories. He decomposes the ideal LIS in multiple modules (Figure 2.2). He also describes the ideal LIS from the functional point of view, discussing desirable functionalities. Those are the following:

- **Information Security**: the LIS must secure information from unauthorised internal and external access. Different user roles should be available with multiple levels of security and information access;
- **Test ordering**: the LIS should handle the ordering provider information, patient information and order information, including real-time feedback on the order. This is an important feature since test ordering systems have the potential to reduce test turnaround time (Westbrook, Georgiou, and Lam 2009);
- **Specimen collection, accessioning and processing**: fundamental to the quality of laboratory results;
- **Analytic phase**: support to the analytic phase lowers the frequency of errors in the clinical laboratory;
- **Result entry and validation**: the LIS should support result entry and validation, helping the laboratory to provide accurate, reproducible and appropriate results;
- **Result reporting**: provides automated result reporting to the laboratory;
- **Notification management**: notifications enable faster responses from the laboratory. The LIS should have multiple tiers of urgency for result notifications;
- **Data mining and cross-sectional reports**: an advanced LIS should explore the collected data by applying data warehousing and data mining techniques improving research processes;
- **Method validation**: it is an important step in the implementation of new assays in clinical laboratories, ensuring the stability of assay systems and compliance with regulatory and accrediting agencies;
- **Quality management**: quality control improves the accuracy and reliability of laboratory results;
- **Administrative and financial issues**: the LIS should support the laboratory management, where administrative and financial issues occur.

2.3.1 History

In the early development of laboratory systems, the terms LIS and LIMS had different applicabilities and functionalities. LISs were installed in hospital laboratories and commercial

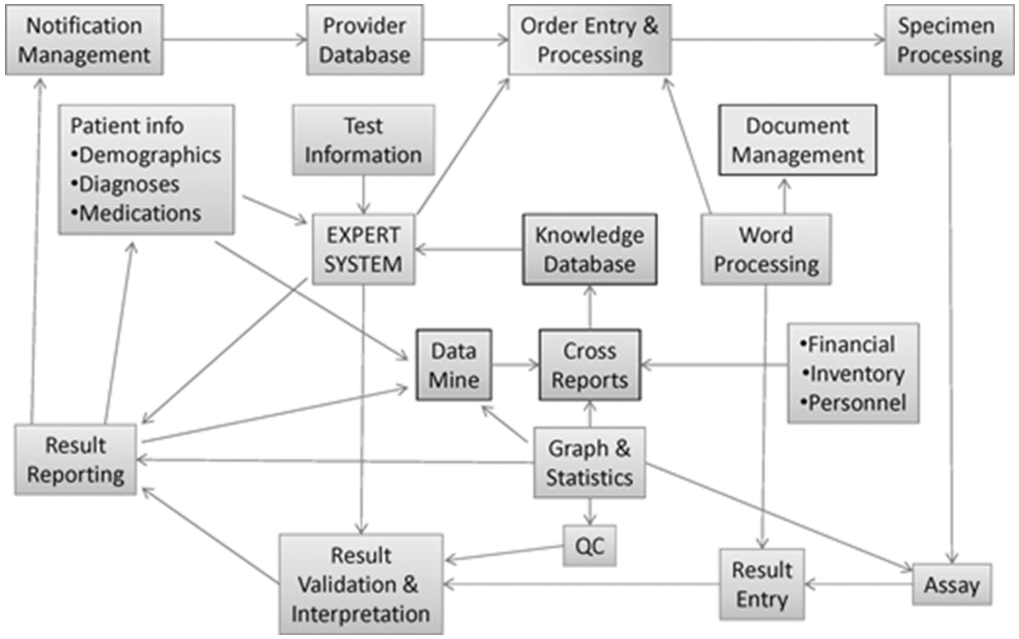


Figure 2.2: Ideal LIS decomposed in modules (Sepulveda and D. S. Young 2013)

reference laboratories, while LIMSs where installed in an industry laboratory or pharmaceutical research laboratories. LISs were designed to report test results for individual patients and satisfy the criteria of hospital accreditation agencies. While LIMSs were designed to report results for batches of samples and satisfy good manufacturing practices (Friedman 2008).

This distinction between a LIS and a LIMS faded over the years as companies started to build single solutions with both systems functionalities. Section 2.4 discusses some examples of these solutions.

2.4 Information System Solutions for COVID-19

The following sections analyse a set of possible market solutions to the problem. In the end, the solutions are compared and conclusions are taken on how they apply to the required functionalities of the current problem.

2.4.1 Belgian LIMS for COVID-19

Weemaes et al. (2020a) developed and implemented a LIMS to manage test ordering, registration, sample flow and result reporting during the COVID-19 pandemic. This LIMS was a response to the increased demand for laboratory testing during the pandemic. Such increase created a strain in the laboratory staff that had to deal with large numbers of samples every day.

They detected that most of the laboratory workforce was doing administrative tasks instead of helping with analytical ones Figure 2.3. The LIMS implementation was able to reduce this administrative burden by streamlining data flows. They implemented the following functionalities to tackle this problem:

- Digital notifications;
- Sample registration;
- Automated scripted triaging;
- Real-time sample tracking;
- Automated validation;
- Automated reporting;
- Daily summary statistics;
- Statistical flagging of outliers.

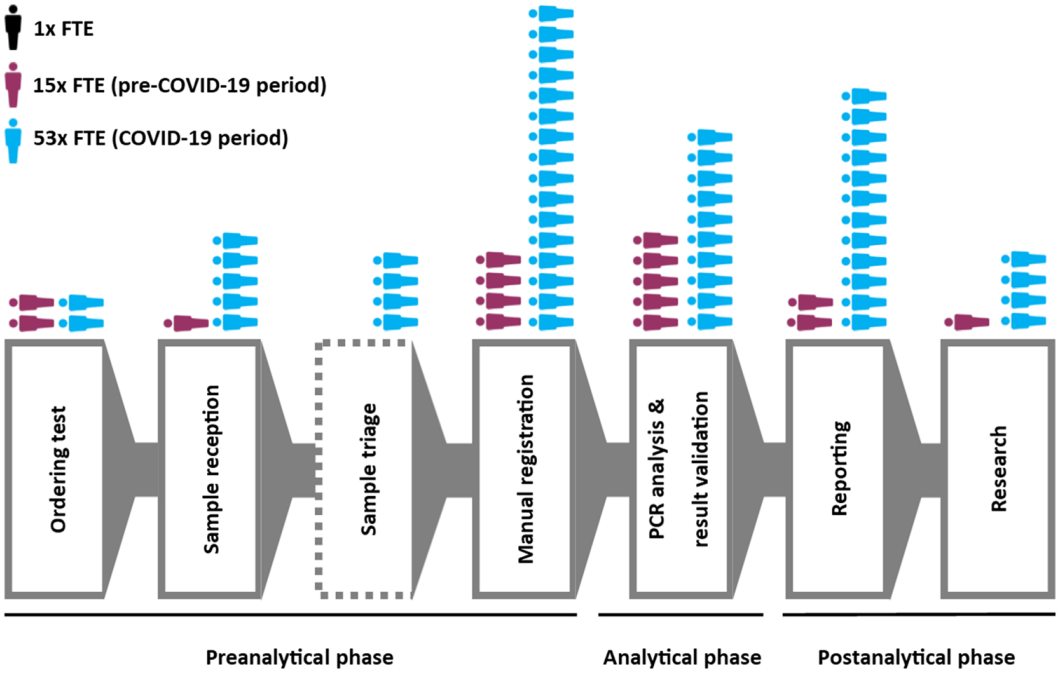


Figure 2.3: Graphical representation of key bottlenecks in the laboratory COVID-19 sample flow (Weemaes et al. 2020a)

Apart from the administrative burden reduction, the LIMS solution also facilitates scientific research. Weemaes et al. (2020a) developed a separate database that recorded all the result parameters in the COVID-19 RT-PCR tests. Along with a standalone data mining application that allowed researchers to visualise the data and identify possible correlations. This tool alleviated the burden on Information Technology (IT) staff and accelerated research.

2.4.2 Thermo Scientific SampleManager LIMS Software

Thermo Fisher Scientific provides, among other products, software solutions in the healthcare area (*SampleManager LIMS Software* n.d.). One of the solutions that they provide is the SampleManager LIMS Software. This LIMS, also known as LIS, manages laboratory data, procedural workflows and integrations with enterprise systems and instruments. Some of the capabilities of this solution are:

- Configurable workflows;

- Mobile application;
- Dashboards and data visualisation;
- Reporting and data analytics;
- Electronic lab notebook support;
- Sample management, accessioning and tracking;
- Instrument management;
- Inventory management.

The SampleManager LIMS can be deployed on the cloud or be self-hosted. Thermo Fisher Scientific has not provided the pricing information since it varies for each client and has to be negotiated.

2.4.3 LabWare LIMS

The LabWare LIMS is a solution for laboratory information management developed by LabWare (*Automate Your Laboratory with the Global Leader for LIMS and ELN* n.d.). This solution has the following functionalities:

- Configurable workflows;
- Mobile application;
- Dashboards and data visualisation;
- Reporting and data analytics;
- Electronic lab notebook support;
- Sample management and tracking;
- Instrument management;
- Inventory management;
- Document management;
- Billing management.

The LabWare LIMS can be deployed on the cloud or on-premises. LabWare does not provide information about the pricing of this solution since it differs for each client.

2.4.4 CrelioHealth

CrelioHealth, formerly know as LiveHealth, is a LIMS solution (*Lab Software for Medical Diagnostics | CrelioHealth LIMS* n.d.). This solution provides the following functionalities:

- Configurable workflows;
- Mobile application;
- Dashboards and data visualisation;
- Reporting and data analytics;
- Electronic lab notebook support;

- Sample management and tracking;
- Instrument management;
- Inventory management;
- Document management;
- Billing management.

CrelioHealth also supports laboratories in COVID-19 testing (*COVID Response* n.d.). The LIMS offers a preconfigured COVID-19 testing workflow. This workflow supports automated and instant reporting, sample registration and automated sample result registration.

CrelioHealth is a paid cloud solution whose price starts at 150$ per month but can go up to 2 200$ per month or 1.5% of the client revenue depending on the selected package.

2.4.5 Solution comparison

Table 2.1 compares the solutions analysed and how they apply to the required functionalities of the current problem.

Table 2.1: Comparison of the identified solutions.

Functionality	**Belgian LIMS**	**Sample Manager LIMS**	**LabWare LIMS**	**CrelioHealth**
Data importation	Yes	No	No	No
RT-PCR COVID-19 test results importation	Yes	No	No	Yes
Report generation	Yes	Yes	Yes	Yes
Authentication and authorisation	Yes	Yes	Yes	Yes
User management	Yes	Yes	Yes	Yes
Sample flow	Yes	Yes	Yes	Yes

Based on the table, it is possible to observe that both Thermo Scientific SampleManager LIMS and LabWare LIMS do not fully support the required functionalities. They lack RT-PCR COVID-19 test results importation and data importation for samples and custom forms.

CrelioHealth, on the other hand, meets almost every criteria since only the data importation for samples and custom forms is missing. However, this is a paid product.

The only solution that supports all the required functionalities is the Belgian LIMS for COVID-19. Only a minor adjustment would have to be made so that the system could support custom forms importation. However, this solution is not open-source and it is not available on the market.

2.5 Conclusion

Healthcare and informatics have a long history together. The idea of computer usage as an aid to medical diagnosis and decision-support has existed for a long time. Information

systems have become crucial to improve medical processes. They can collect, process and store data, lower communication costs inside an organisation and improve supply chain coordination. During the ongoing pandemic, these systems enabled organisations to work from anywhere without significantly impacting the profit.

When it comes to healthcare, information systems lead to higher administrative efficiency, reduce medical errors and improve coordination. However, not every information system is successful, and some are abandoned or do not fulfil the desirable outcomes. The design-reality gap identifies the hard-soft and design-implementation gaps as major archetypes for an information system's failure.

As for the LISs, they are essential to manage the flow of information in laboratories. From a functional point of view, they should support information security, test ordering, specimen collection, accessioning and processing, result entry, validation and reporting, among others. During market research for possible solutions to the current problem, it was concluded that most of the systems support most of the functionalities of an ideal LIS but fail to meet the new requirements of COVID-19 testing. Only the Belgian LIMS supports all the required functionalities; however, the solution is not open-source and is not available on the market. Based on that, it is possible to conclude that a new LIS is necessary, and it would be a market innovation.

Chapter 3

Value Analysis

This chapter analyses the value of the current dissertation. It applies the NCD model, the value proposition canvas and the QFD to this dissertation.

3.1 The New Concept Development Model

The NCD model is a theoretical construct for innovation's Fuzzy Front End (FFE). NCD provides a common framework and language for the front end activities. Figure 3.1 demonstrates that the NCD model consists of three essential parts:

- Five key elements represented in the inner area;
- The engine that drives the five elements through leadership and culture of the organisation;
- Influencing factors that can affect the innovation process (Koen et al. 2001).

The NCD model fits the current project and helps in understanding its value better. In the following subsections, each key element is applied to the project.

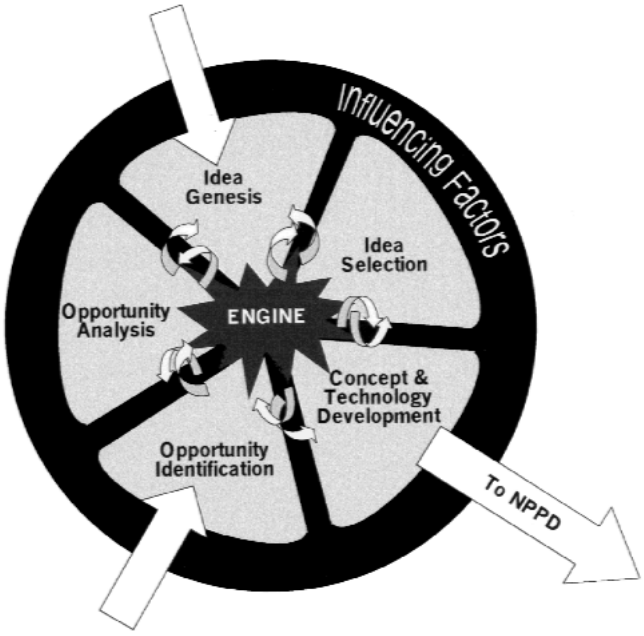


Figure 3.1: NCD model (Koen et al. 2001)

3.1.1 Opportunity Identification

The COVID-19 pandemic hit the world by surprise. Given the nature of the virus, it was highly contagious and difficult to track. The only way to diagnose COVID-19 was through a test, and the RT-PCR tests proved to be the most precise.

According to Dowdy and D'Souza (2020), a high positive percentage of tests indicates high infection rates. Following the World Health Organization (WHO) recommendations, Dowdy and D'Souza (2020) say that a 5% positive percentage is the threshold, and countries should aim to be below it. The positive percentage should decrease by reducing the amount of COVID-19 transmission or increasing the number of tests. The two are related and more tests leads to lower amounts of transmission.

On the first 26 days of January 2021, Portugal reached an average of 52 thousand daily tests (Diário de Notícias 2021). During that period, 240 200 of the tests were positive (Direção-Geral da Saúde 2021), leading to an average of 9 238 new infections per day. This results in 17.8% positive tests indicating that Portugal should be testing more.

PORTIC is responsible for testing the P.Porto community and various health centres of the north. They deal with a significant amount of tests per day and reached maximum capacity. The biggest problem is the amount of information to be handled. The staff wastes a lot of their time managing information between health centres, the P.Porto community and ARSN. If the staff could manage the information faster and consistently, PORTIC would be able to increase the number of daily tests. This problem results in an opportunity to build an information system for PORTIC.

3.1.2 Opportunity Analysis

Even though the COVID-19 pandemic is recent, there are advances in the state of the art for information systems. Reeves et al. (2020) published an article about an information system focused on the outbreak management of COVID-19. Among other functionalities, this system supports reporting and analytics for tests. However, the system does not support data importation from external sources. One other information system to consider is from an article published by Weemaes et al. (2020b). This system manages test registration, sample flow and result reporting. It also supports data importation from external sources. However, due to the specific external data sources that PORTIC requires, it would be necessary to implement them. This system also lacks authentication and authorisation, which is an important requirement for PORTIC.

After the opportunity analysis, it is possible to conclude that building a new information system is worth it. There are few information systems for COVID-19 testing, and the ones that exist do not support important requirements for PORTIC.

3.1.3 Idea Genesis

After some discussion about the new information system, some ideas about possible functionalities arose. Those are the following:

- Data importation from different external sources;
- Importation of the RT-PCR test results;
- Report Generation;

- Authentication and authorisation;
- Users management;
- Stock management;
- Application of data mining techniques to the collected data.

3.1.4 Idea Selection

Due to time concerns and value retrieved to PORTIC, some functionalities were not selected to be implemented. The list of chosen functionalities to implement is the following:

- Data importation from different external sources;
- Importation of the RT-PCR test results;
- Report Generation;
- Authentication and authorisation;
- Users management.

3.1.5 Concept and Technology Development

With the development of an information system, the PORTIC laboratory staff will manage the information faster and with data consistency. This system will help the staff focus on sample analysis and test result determination. It will also enable the laboratory to increase the number of daily tests made.

3.2 Value, Value for the Customer and Perceived Value

This section defines the concepts of value, value for the customer and perceived value. It also applies these concepts in the context of this dissertation.

3.2.1 Value

According to Nicola, E. P. Ferreira, and J. J. Ferreira (2012), "Value has been defined in different theoretical contexts as need, desire, interest, standard/criteria, beliefs, attitudes, and preferences". Value varies according to different perceptions. A customer and a supplier have different perceptions of the value of a product. As such, value is subjective, and the perceived value of a product can differ from the value for the customer.

3.2.2 Value for the Customer

Value for the customer is often used in the marketing literature to represent what the customer perceives or receives from the product. It can be represented through multiple perspectives and varies between different customers (Woodall 2003).

3.2.3 Perceived Value

Zeithaml (1988) defined perceived value has "the consumer's overall assessment of the utility of a product based on perceptions of what is received and what is given". What a consumer

receives and gives varies across multiple consumers, indicating that value is a trade-off of benefits and sacrifices.

Table 3.1 applies these definitions of value, value for the customer and perceived value by comparing the benefits and sacrifices of a consumer to the current dissertation's product.

Table 3.1: Benefits and sacrifices.

Benefits	**Sacrifices**
Easier information management	Added infrastructure costs
Faster data treatment	The dependency of internet connection
Data security and consistency	The learning curve of the software usage
Allow more staff to manage part of the information	
Faster test results	

3.3 Value Proposition

A value proposition is an overall view of the value that a company's products or services offer to the customers. This statement is part of the marketing strategy of a company. Targets customers who will benefit from the product and states the reason why it is unique. A value proposition answers questions as: "What is your product?"; "Who is your target customer?"; "What value do you provide?"; "Why is your product unique?". As such, the current dissertation's product's value propositions is the following:

> **Test better, test faster**
>
> *Developed to help in the fight against a global pandemic, this information system will allow your laboratory faster test results.*

3.4 Value Proposition Canvas

Osterwalder et al. (2014) created the value proposition Canvas, a framework for product or service positioning to meet customer needs. The application of this Canvas should target a single customer segment. Consists of two blocks, customer profile and value map. The customer profile divides into three sections: gains, expected benefits for the customers; pains, risks and negative experiences; customer jobs: functional, emotional and social tasks that the customers are trying to execute. The value map also divides into three sections: gain creators, how the product or service creates customer gains; pain relievers, how the product or service eliminates or reduces customer pains; products and services: the products and services that create value for the customer. The value map and customer profile aim to reach fit. A value proposition must create essential gains, alleviate extreme pains and address important jobs, to achieve fit.

Figure 3.2 applies this Canvas to the current project.

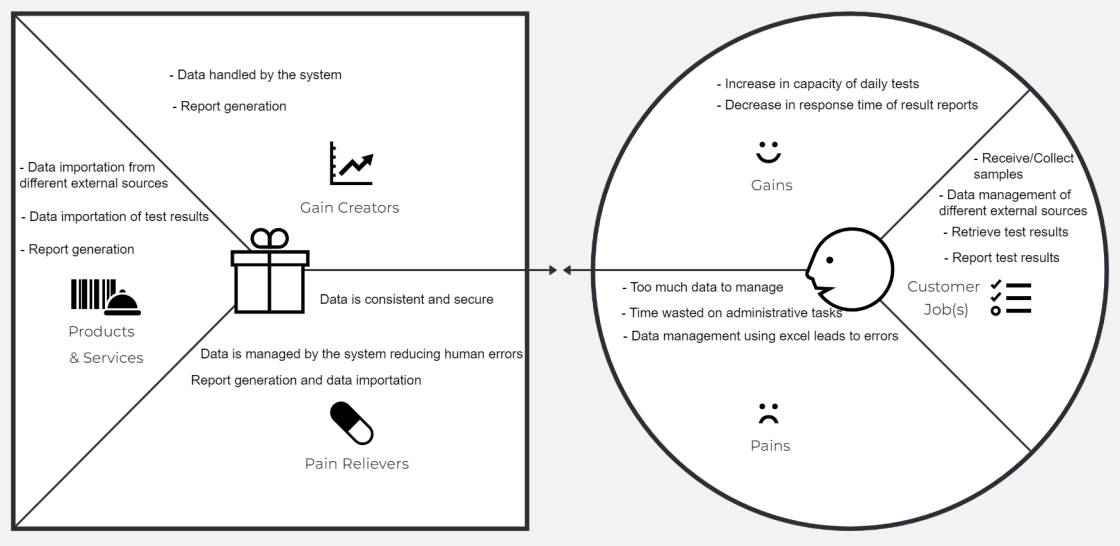
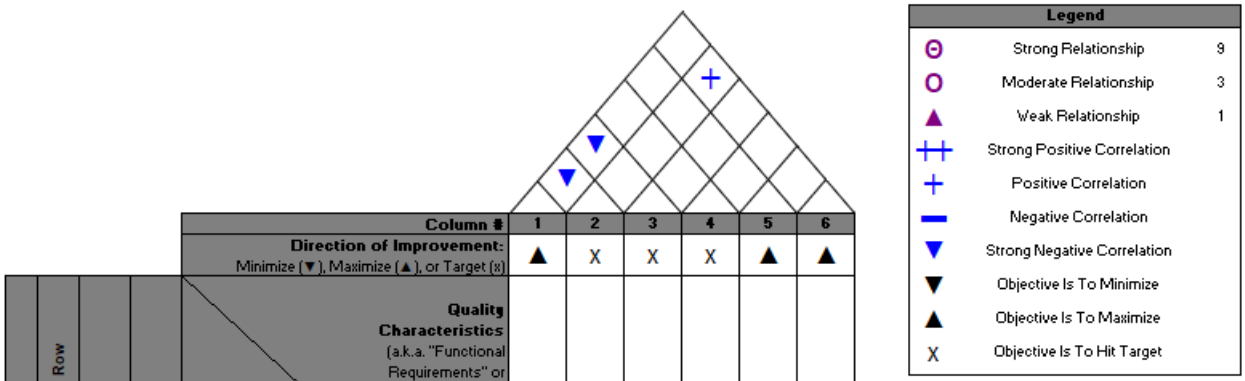


Figure 3.2: Value Proposition Canvas

3.5 Quality Function Deployment

QFD is a concept that "ensures that customer requirements are integrated into new products as early as the design stage" (Zairi and Youssef 1995). The house of quality, part of QFD, serves to translate the customer requirements to engineering characteristics (Hauser and Clausing 1988).

Figure 3.3 applies the house of quality to the current project.



Legend		
⊖	Strong Relationship	9
O	Moderate Relationship	3
▲	Weak Relationship	1
++	Strong Positive Correlation	
+	Positive Correlation	
−	Negative Correlation	
▼	Strong Negative Correlation	
▼	Objective Is To Minimize	
▲	Objective Is To Maximize	
X	Objective Is To Hit Target	

Row #	Max Relationship Value in Row	Relative Weight	Weight / Importance	Demanded Quality (a.k.a. "Customer Requirements" or "Whats") / Quality Characteristics (a.k.a. "Functional Requirements" or "Hows")	1 Encryption	2 File Importation	3 File exportation	4 Email Service	5 User Roles and claims	6 Code generation
				Direction of Improvement: Minimize (▼), Maximize (▲), or Target (x)	▲	X	X	X	▲	▲
1	9	14.9	5.0	Import samples from health centres	⊖	⊖				O
2	9	10.4	3.5	Import submissions from IPP form	⊖	⊖				
3	3	14.9	5.0	Register samples from IPP	O					O
4	9	14.9	5.0	Import RT-PCR tests results		⊖				
5	9	6.0	2.0	Generate reports for health centres tests			⊖	O		
6	9	4.5	1.5	Generate individual reports for IPP tests	O		⊖	O		
7	9	11.9	4.0	Automatic generation of laboratory code		▲				⊖
8	9	13.4	4.5	Data security	⊖	▲	▲	▲		
9	9	6.0	2.0	Authentication and authorization		▲	▲	O	⊖	
10	9	3.0	1.0	Authorization management					⊖	
				Max Relationship Value in Column	9	9	9	3	9	9
				Weight / Importance	407.5	394.0	113.4	62.7	80.6	197.0
				Relative Weight	32.5	31.4	9.0	5.0	6.4	15.7

Figure 3.3: House of quality

Chapter 4

Analysis and Design

This chapter describes the functional and non-functional requirements for the system to develop. It considers relevant technologies for its development, identifying the technologies to use. Lastly, it details the architectural design of the solution, reviewing possible alternatives.

4.1 Requirements Engineering

This section defines and documents the requirements for the system to develop. During the process of requirements elicitation, the author applied various elicitation techniques, namely:

- **Interviews:** used several times to understand what each user needs;
- **Workshops:** used to define requirements with multiple stakeholders when their vision of the product collided;
- **Observations:** used once to understand the entire process of the laboratory and how the users perform their tasks;
- **Document analysis:** required when possible to uncover new information;
- **System interface analysis:** used once to understand how the RT-PCR works and how it provides the test results.

A QFD is useful to translate the needs of the customers into technical requirements for the software. Section 3.5 applies this concept.

4.1.1 Actors

An actor is a role played by an external user or system that interacts with the system, usually through a use case. Table 4.1 specifies and describes the actors of the current system.

Table 4.1: Actors

Actor	Description
Laboratory administrator	Responsible for the management and administration of the laboratory. This actor must have access to all the functionalities, including user management, where he manages users authorisations.
Sample collector	Responsible for the sample collection. This actor must register the samples collected.
Tester	Responsible for analysing the RT-PCR test results. This actor must import the COVID-19 test results.

4.1.2 Domain Model

Evans (2015) defines a domain as "a sphere of knowledge, influence, or activity. The subject area to which the user applies a program is the domain of the software." and model as "a system of abstractions that describes selected aspects of a domain and can be used to solve problems related to that domain.". Figure 4.1 demonstrates the domain model defined for this problem.

The following concept definitions help to understand the domain model defined:

- **Sample**: represents the sample to analyse and test. It can be collected by the laboratory or sent by health centres, and both a COVID code and a laboratory code identify it;
- **COVID code**: identifies a sample. It is composed of an organisation identifier and an incremental number;
- **Laboratory code**: identifies a sample. It is composed of a letter that represents the day of the week and an incremental number;
- **Sample type**: represents the sample type, which can be Health centre, if sent by a health centre or P.Porto, if collected from the P.Porto community;
- **Test**: represents a COVID-19 test made to a sample;
- **Result**: represents the result of the COVID-19 test, which can be negative, positive or inconclusive;
- **Submission**: represents a form submission from the P.Porto community. It includes the information required on the form;
- **Person**: represents the person that submitted the form submission;
- **Gender**: represents the gender of the person, which can be male, female or other;
- **Location**: represents the location address of the person. It is consists of the city and the address;
- **Symptom**: represents the symptom that the person had;
- **Symptom type**: represents the type of symptom, which can be respiratory, gastrointestinal or other.

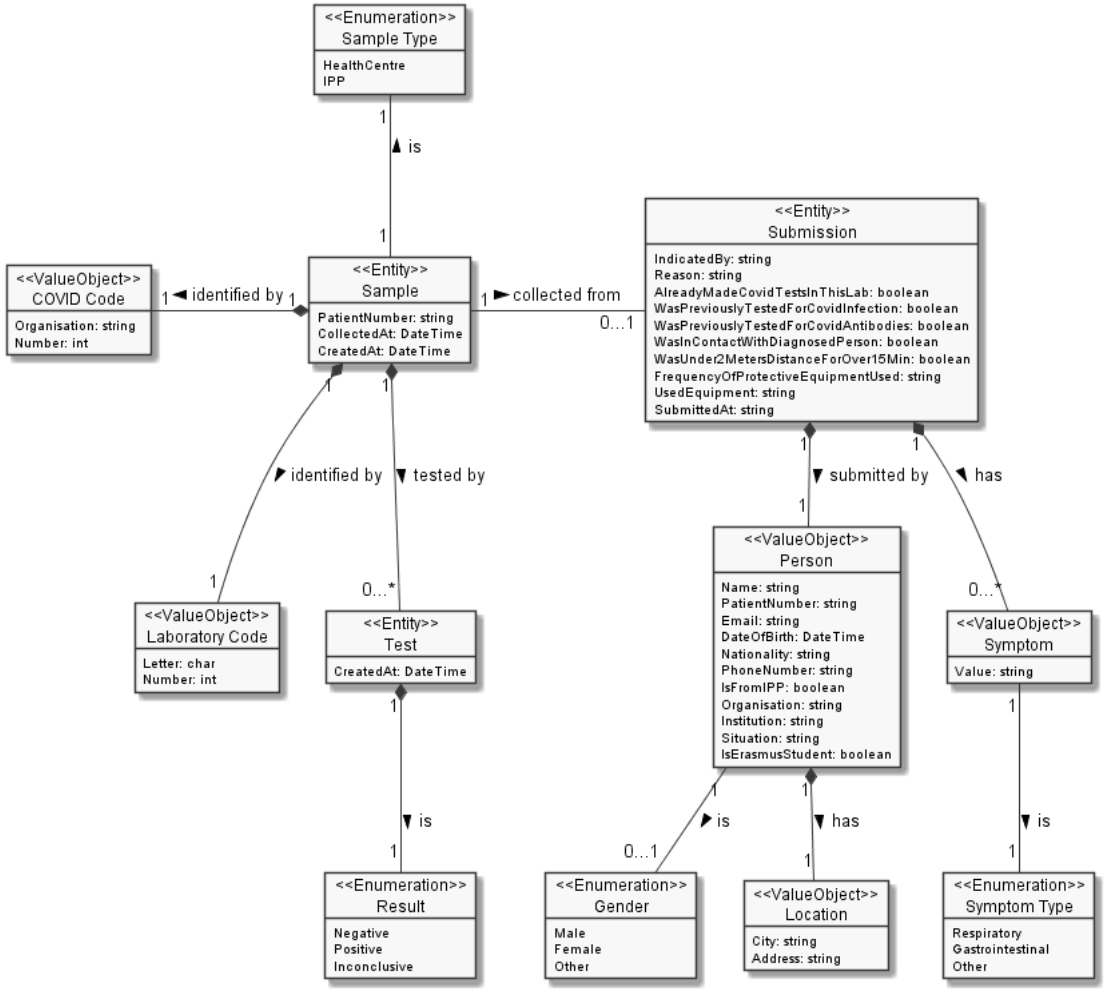


Figure 4.1: Domain Model

4.1.3 Functional Requirements

Functional requirements "describe what the system or software must do. (...) Functional requirements are sometimes called behavioural or operational requirements because they specify the inputs (stimuli) to the system, the outputs (responses) from the system, and behavioural relationships between them." (R. R. Young 2004).

At the start of the project, the functional requirements presented in Table 4.2 were identified. However, during the project development, new requirements emerged, and others changed. REQ-05a and REQ-06a ceased to exist. Instead of generating reports by health centre and for each P.Porto submission, REQ-07 and REQ-08 fill those requirements. The search samples functionality allows the user to filter by date and by health centre. At the same time, the export samples enables the user to export that filtered data and select what data to export. The rest of the initial functional requirements remain unchanged, and only new ones were added. Table 4.3 shows the final functional requirements list.

Table 4.2: Initial Functional Requirements

Identifier	Description	Actors
REQ-01a	Import Samples from health centres	Laboratory Administrator
REQ-02a	Import P.Porto submissions	Laboratory Administrator
REQ-03a	Register a new sample	Laboratory Administrator; Sample Collector
REQ-04a	Import COVID-19 test results	Laboratory Administrator; Tester
REQ-05a	Generate test reports by health centre	Laboratory Administrator
REQ-06a	Generate individual test reports for P.Porto submissions	Laboratory Administrator
REQ-07a	Search samples	Laboratory Administrator; Sample Collector
REQ-08a	Export samples	Laboratory Administrator; Sample Collector

Table 4.3: Final Functional Requirements

Identifier	Description	Actors
REQ-01	Import Samples from health centres	Laboratory Administrator
REQ-02	Import P.Porto submissions	Laboratory Administrator
REQ-03	Register a new sample	Laboratory Administrator; Sample Collector
REQ-04	Import COVID-19 test results	Laboratory Administrator; Tester
REQ-05	Search samples	Laboratory Administrator; Sample Collector
REQ-06	Export samples	Laboratory Administrator; Sample Collector
REQ-07	Update sample	Laboratory Administrator; Sample Collector
REQ-08	Delete sample	Laboratory Administrator; Sample Collector
REQ-09	Get sample details	Laboratory Administrator; Sample Collector
REQ-10	Register a new P.Porto submission	Laboratory Administrator
REQ-11	Update P.Porto submission	Laboratory Administrator
REQ-12	Delete P.Porto submission	Laboratory Administrator
REQ-13	Associate sample to P.Porto submission	Laboratory Administrator; Sample Collector
REQ-14	Decouple sample from P.Porto submission	Laboratory Administrator; Sample Collector
REQ-15	Search submissions	Laboratory Administrator; Sample Collector
REQ-16	Get submission details	Laboratory Administrator; Sample Collector

REQ-01: Import samples from health centres

Description

The Laboratory Administrator is responsible for the importation of samples from the health centres. The information is imported through an excel file that the health centres send to the laboratory. The file represents the samples that the health centre collected and sent for the laboratory to test. It contains the COVID code, the patient number and the collection date of the sample. The Laboratory Administrator indicates the header of the file and the columns of each information since different health centres send different files.

Preconditions

- the user is authenticated;
- the user has the Laboratory Administrator role;
- the samples must have been received by the laboratory.

Successful event flow

1. The Laboratory Administrator initiates the health centre samples import.
2. The system asks for the file, the file header and the columns of the COVID code, collection date and patient number.
3. The Laboratory Administrator chooses the file to import and indicates the respective header and columns.
4. The system validates the format of the file and imports the samples successfully with a generated laboratory code.

Alternative event flows

4a. The system detects an error on a line and alerts the user that the importation stopped on that line, importing the previous ones.

4b. The system detects a not supported format and alerts the user.

System Sequence Diagram

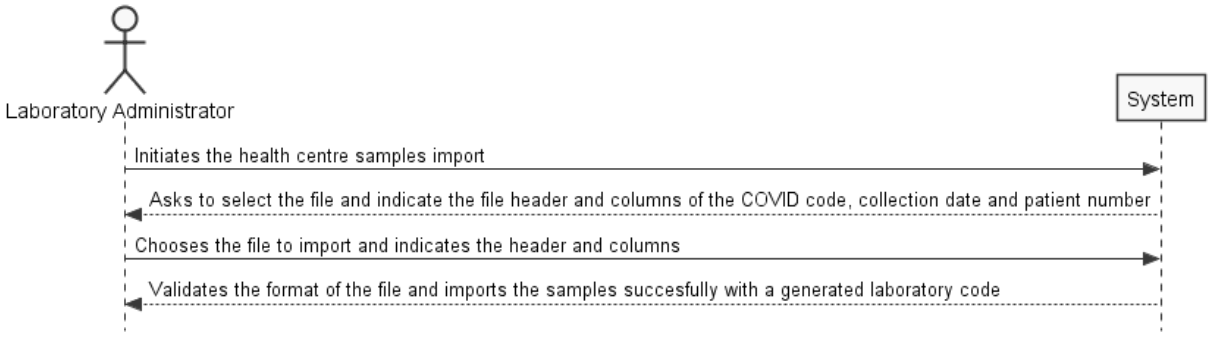


Figure 4.2: REQ-01: System Sequence Diagram

Postconditions

- the samples are recorded in the system;
- the samples have a laboratory code generated by the system.

Validations

- the COVID code, the patient number and the collection date are mandatory.

REQ-05a: Generate test reports by a health centre

Description

The Laboratory Administrator is responsible for the generation of test reports by a health centre. The reports are exported in an excel file. They represent the samples sent by the health centre, along with the COVID-19 test results. And they must have the collection date, the COVID code, the patient number and the test result of each sample.

Preconditions

- the user is authenticated;
- the user has the Laboratory Administrator role;

- the samples must have a test result registered.

Successful event flow

1. The Laboratory Administrator initiates the test report generation of a health centre.
2. The system asks for a health centre identifier and a day.
3. The Laboratory Administrator indicates the health centre and the day.
4. The system generates and exports the report for all the samples of the selected day and health centre.

Alternative event flows

4a. The system detects that there are no samples to generate a report and alerts the user.

System Sequence Diagram

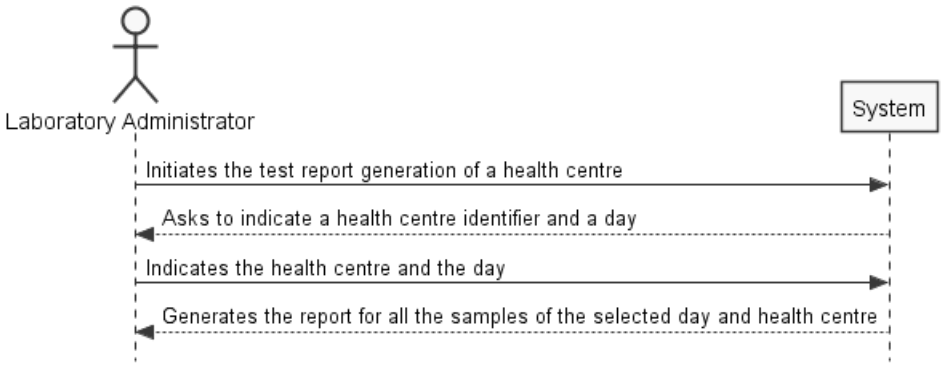


Figure 4.3: REQ-05a: System Sequence Diagram

Postconditions

- the report is exported in an excel file.

Validations

- the day and the health centre are mandatory.

4.1.4 Non-functional Requirements

Non-functional requirements are attributes of or constraints on a system (Glinz 2007). Table 4.4 classifies the non-functional requirements according to the FURPS+ classification model developed at Hewlett-Packard.

Table 4.4: Non-functional requirements

Identifier	Description	Classification
RNF01	The system must have an authentication mechanism	Functional
RNF02	The system must have an authorisation mechanism	Functional
RNF03	The technologies used must be open-source or from Microsoft	Implementation
RNF04	The personal data stored must be encrypted	Security
RNF05	The application must support access from Chrome and Firefox	Design
RNF06	The system must be easily maintained by other P.Porto personnel	Supportability
RNF07	The system must be extensible	Supportability
RNF08	The User Interface (UI) must be intuitive	Usability
RNF09	The UI must be responsive	Usability
RNF10	The system must be easily tested	Supportability
RNF11	The database must be SQL Server	Design

4.2 Relevant Technologies

RNF05 indicates that the solution must be a web application since it has to be accessed by a web browser. On the other hand, RNF03 declares that the used technologies must be open-source or from Microsoft. As such, this section explores relevant technologies that satisfy these requirements for the web application and the database.

For the web application, the following solutions satisfy RNF05 and RNF03:

- **Node.js**;
- **Ruby on Rails**;
- **ASP.NET Core**.

All of these technologies are open-source and support web development. They execute on Windows and Linux servers and can are deployable on the cloud or on-premises. They all satisfy RNF01 and RNF02 since they all support authentication and authorisation. As such, all of them are suitable options for the development of this solution. Considering the urgency of this solution (Section 1.1 and 1.2) and the author experience, ASP.NET Core is the technology chosen.

ASP.NET Core offers ASP.NET Core MVC for web development (Microsoft 2020b). It utilises Razor Pages for the user interface and ASP.NET Identity to handle authentication and authorisation (Microsoft 2019).

For the database, RNF11 imposes the use of SQL Server as the database for the web application.

4.3 Architectural Design

Software architecture is "the process of defining a structured solution that meets all of the technical and operational requirements, while optimising common quality attributes" (Microsoft 2009). It comprises a set of significant decisions about the organisation of a software system and has a high impact on the solution's success.

4.3.1 Candidate Architectures

This section describes possible architectural solutions for the problem in question. Each alternative is analysed on how it responds to the non-functional requirements, concluding on a solution.

N-Layer architecture

This architectural style divides the application logic into layers. When the application logic is distributed across separate deployment targets, these deployment targets are called tiers. The number of tiers does not have to correspond to the number of layers in an application (Microsoft 2020a).

A layered architecture has advantages in the code organisation. The reusability of common functionalities across the application is one of them. This architecture style also enforces restrictions on layer communication, promoting encapsulation and loose coupling. For instance, a change should only take place in a single layer, and only layers that communicate with it should be affected. This makes the replacement of a functionality implementation easier, both for requirement changes and testing purposes.

A common organisation of the application layers is the 3-layer architecture. It consists of a UI, or presentation, layer, a business logic layer and a data access layer. Figure 4.4 demonstrates this architecture.

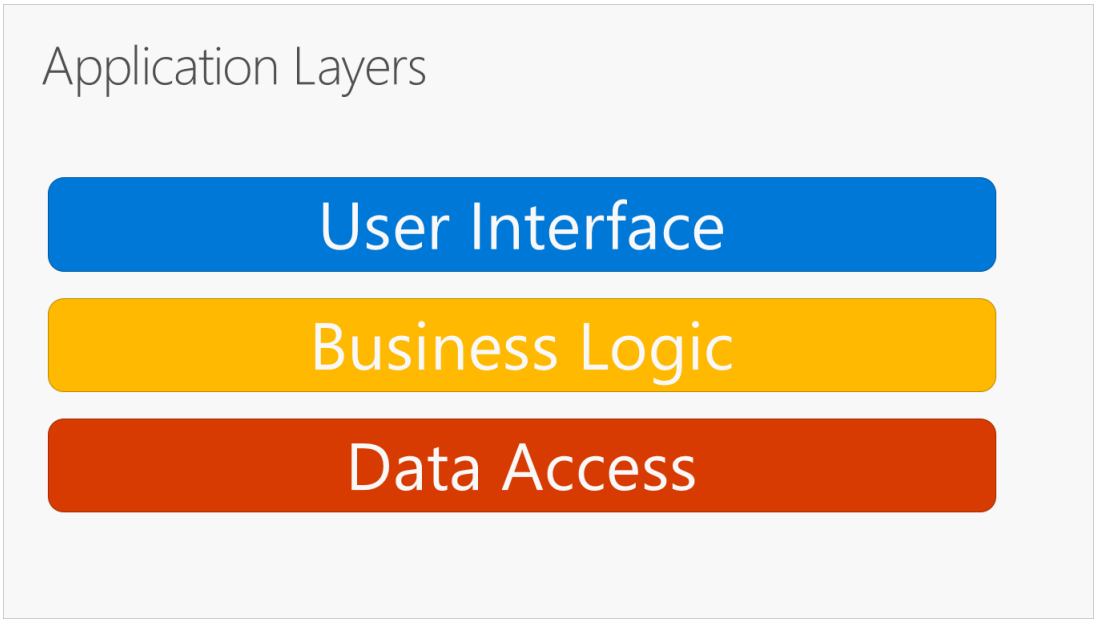


Figure 4.4: 3-Layer architecture (Microsoft 2020a)

In this architecture, the user interacts with the application through the UI layer, which only interacts with the business logic layer. On the other hand, the business logic layer only interacts with the data access layer. This communication cannot be reversed, and it cannot skip layers. In other words, the UI layer cannot directly call the data access layer, nor the business logic layer can directly communicate with the presentation layer.

A problem with this architecture is that each layer depends on the layer below. This makes the business logic layer depend on the data access layer. As such, the business rules cannot be tested without the data access layer or a fake implementation of it. This affects the extensibility and testability of the system, going against the RNF07 and RNF10. For that reason, this architecture style is rejected.

Clean Architecture

Clean architecture's main objective is the separation of concerns. It achieves this objective by splitting the application into multiple layers, similar to the n-layer architecture (Martin 2012). The idea of clean architecture is to make a system:

- **Independent of frameworks**: allows the system to use frameworks as tools and not depend on them. This way, the system is not limited by a framework's constraints.
- **Testable**: the business rules can be tested in isolation and without the dependency of another layer.
- **Independent of UI**: the UI is easily replaced without affecting the rest of the system.
- **Independent of the database**: the database can be replaced without affecting the rest of the system. This includes the business rules which are not coupled to the database.
- **Independent of any external agency**: the business rules are not affected by external agencies.

Figure 4.5 demonstrates an example of clean architecture where the application has the following layers (Microsoft 2020a):

- **UI**: responsible for the implementation of the presentation logic.
- **Infrastructure**: responsible for the implementation of the persistence logic, external systems and frameworks.
- **Application Core**: responsible for the implementation of the business rules and application model.

Clean architecture enforces a dependency rule where dependencies can only point inwards. Since the business rules and application model are the core of the application, the UI and infrastructure depend on it. This dependency inversion is possible through abstractions or interfaces in the application core that are implemented by the infrastructure layer. As such, this architectural style achieves the RNF06, RNF07 and RNF10, since it provides an extensible application, easily tested and maintained. For these reasons, clean architecture is the chosen architectural style for the solution.

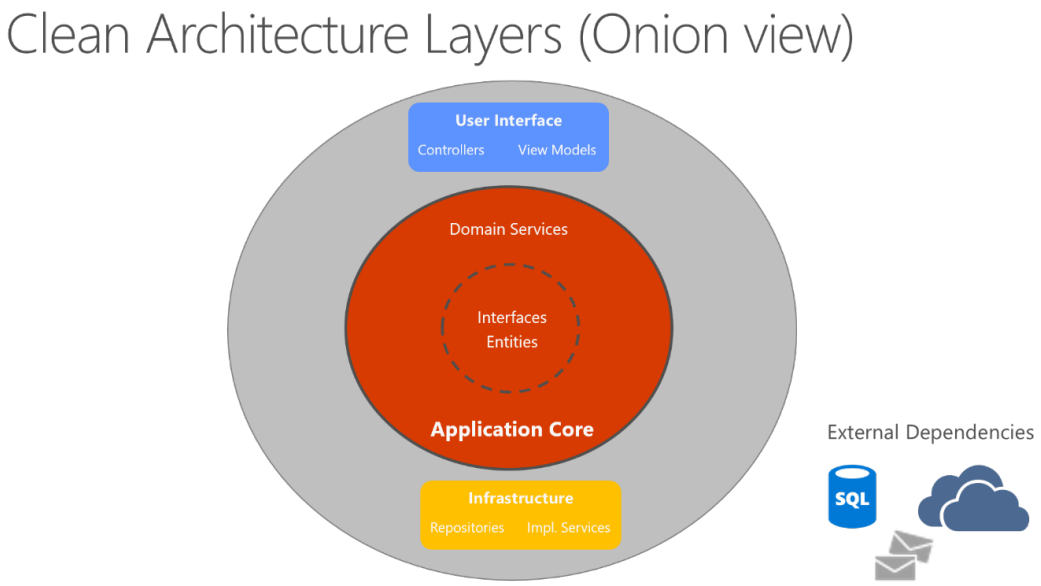


Figure 4.5: Clean architecture (Microsoft 2020a)

4.3.2 Web Application

According to RNF05, the application must be accessed by a web browser, namely Chrome and Firefox. As such, the solution must be a web application. Figure 4.6 demonstrates the component diagram of the solution. It is composed of the following components:

- **Presentation**: represents the UI layer of clean architecture. It is responsible for the UI logic.
- **Core**: represents the application core layer of clean architecture. It is responsible for the business rules of the application.
  - **Application**: represents the domain services of the application. It is a part of the core layer.
  - **Domain**: represents the domain model of the application. It is a part of the core layer.
- **Infrastructure**: represents the infrastructure of clean architecture. It is responsible for the data access logic, external systems logic, authentication, authorisation and frameworks.
- **Database**: represents the database of the application.

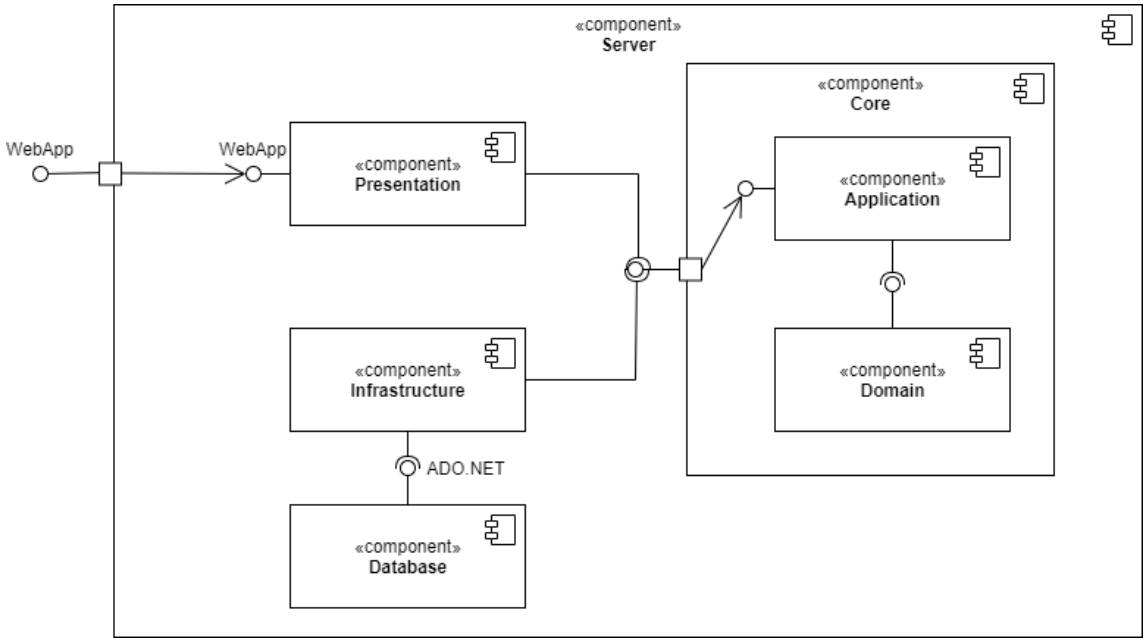


Figure 4.6: Component Diagram

4.3.3 Deployment View

The deployment view illustrates the physical distribution of the system nodes. For this solution, the web application and the database remain on the same server since P.Porto only provides a single server for the application. This server's operating system is Ubuntu 18.04.5 LTS, and the web application runs in Nginx, an open-source HTTP server that satisfies RNF03. The web application connects to the SQL server database directly, and the user communicates with the application through a web browser. Figure 4.7 demonstrates the deployment view of the solution.

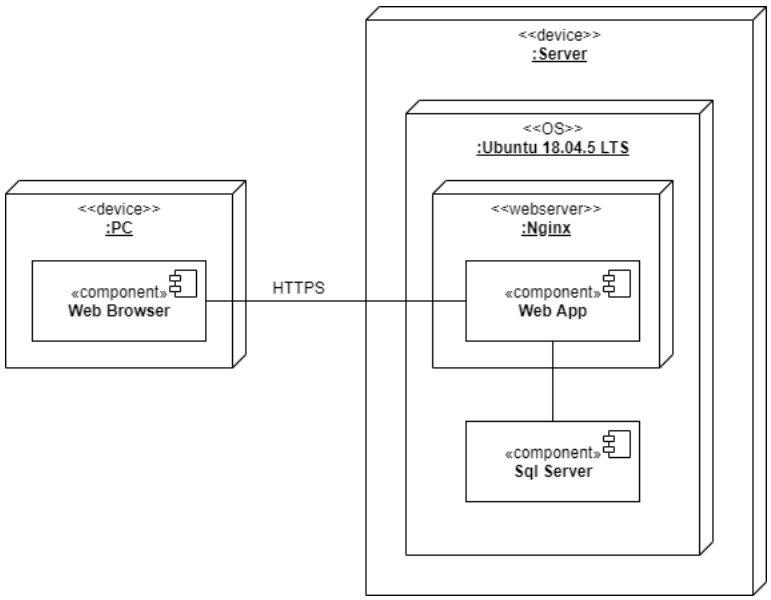


Figure 4.7: Deployment Diagram

4.3.4 Database Model

The database model for this solution splits into two. One model to answer the functional requirements and another to answer RNF01 and RNF02. The model to answer the authentication and authorisation requirements is based on the ASP.NET Identity since it is the Microsoft solution to handle authentication and authorisation. Figures 4.8 and 4.9 illustrate the database models.

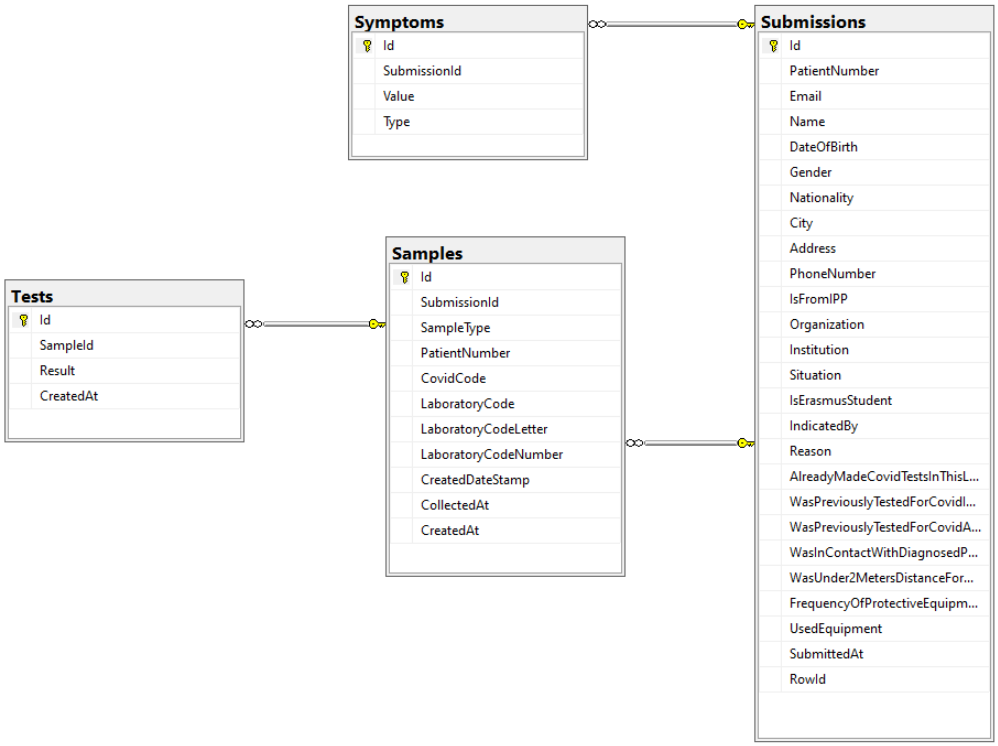


Figure 4.8: Database Model

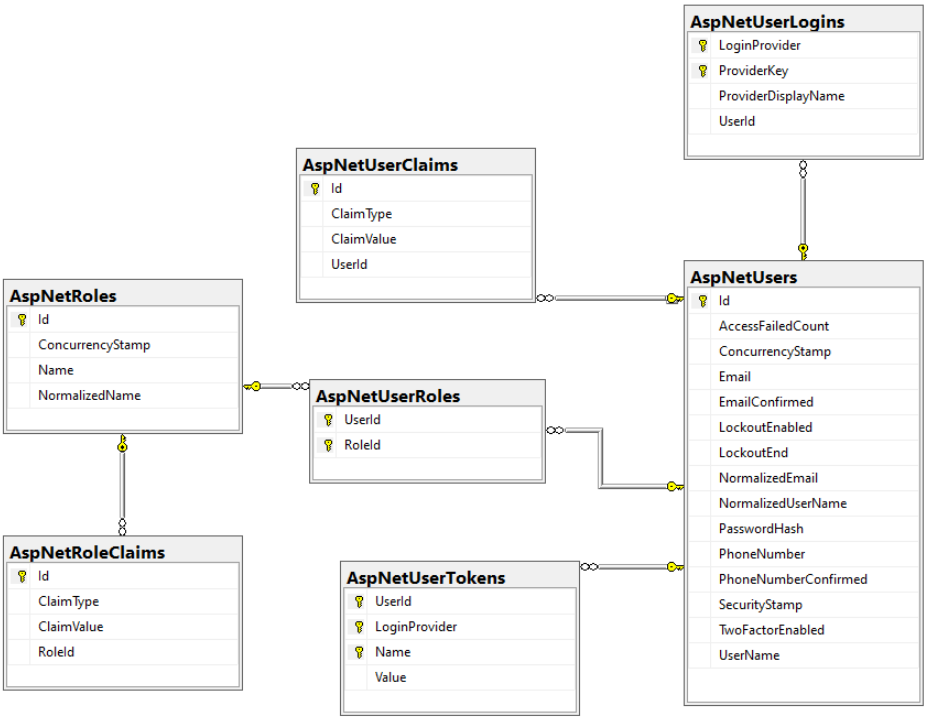


Figure 4.9: Database Identity Model

Chapter 5

Implementation

The focus of this chapter is on the implementation details of the application. It explains how the code was structured, relating it to the layered architecture used. It describes the design patterns adopted and shows examples of their usage. The chapter also describes how the application tackles some functional and non-functional requirements. Finally, it explains how this application was tested.

5.1 Code Structure

Clean architecture is the chosen architecture for this application. Section 4.3.1 shows that the clean architecture splits the application into the Web, Infrastructure and Application Core layers, and Section 4.3.2 relates those layers with the Presentation, Application, Domain and Infrastructure components. This section concentrates on how those layers and components are structured in the source code, using module diagrams (Figure 5.1).

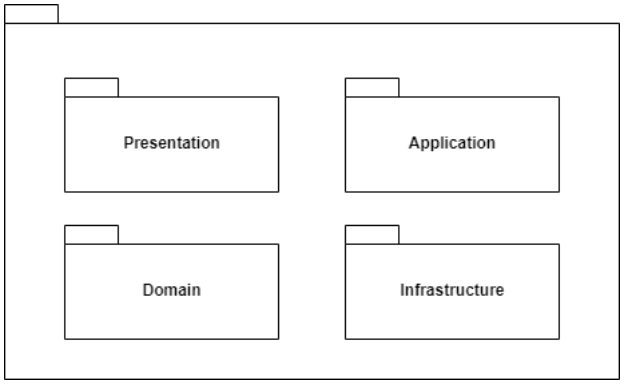


Figure 5.1: Application Modules

5.1.1 Domain Module

The Domain module composes the core layer of the application and represents the business rules and core logic. Figure 5.2 displays the module view of the Domain module, along with the following submodules:

- **Entities:** contains all the entities of the domain model. This is where the core logic of the domain exists in the form of mutable objects;
- **Enums:** contains all the enumerators of the domain model;

- **ValueObjects:** contains the value objects of the domain model. This is where the core logic of the domain exists in the form of immutable objects.

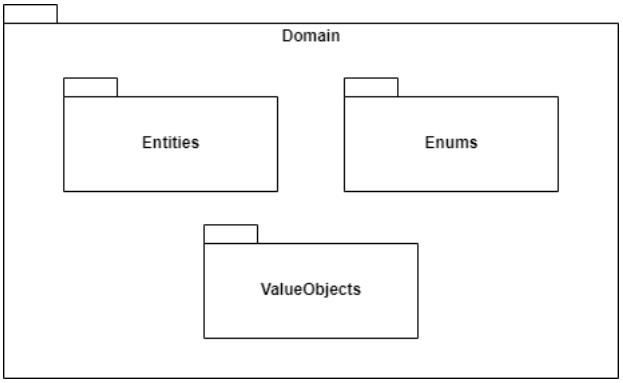


Figure 5.2: Domain Module

5.1.2 Application Module

The Application module completes the core layer of the application alongside the Domain module. This module contains logic that does not fit the domain model, typically translating directly to a functional requirement. The Submissions and Samples modules are submodules of the application module and represent entities of the domain model. Both of these smaller modules have:

- **Commands:** contains all the commands of the application. It represents the command part of the Command Query Responsibility Segregation (CQRS) pattern, in detail in Section 5.2.1;
- **Queries:** contains all the queries of the application. It represents the query part of the CQRS pattern.

The Application module also includes a Common module; which splits into the following modules:

- **Behaviours:** has all the behaviours of the commands and queries. These are part of the Mediator pattern, in detail in Section 5.2.2;
- **Exceptions:** has the custom exceptions used in the commands and queries;
- **Interfaces:** has the interfaces required for the commands and queries. These interfaces serve as an abstraction to the actual implementation, following the Inversion of Control principle. They are implemented in the Infrastructure module;
- **Models:** has generic models for the commands and queries.

Figure 5.3 represents the Application module.

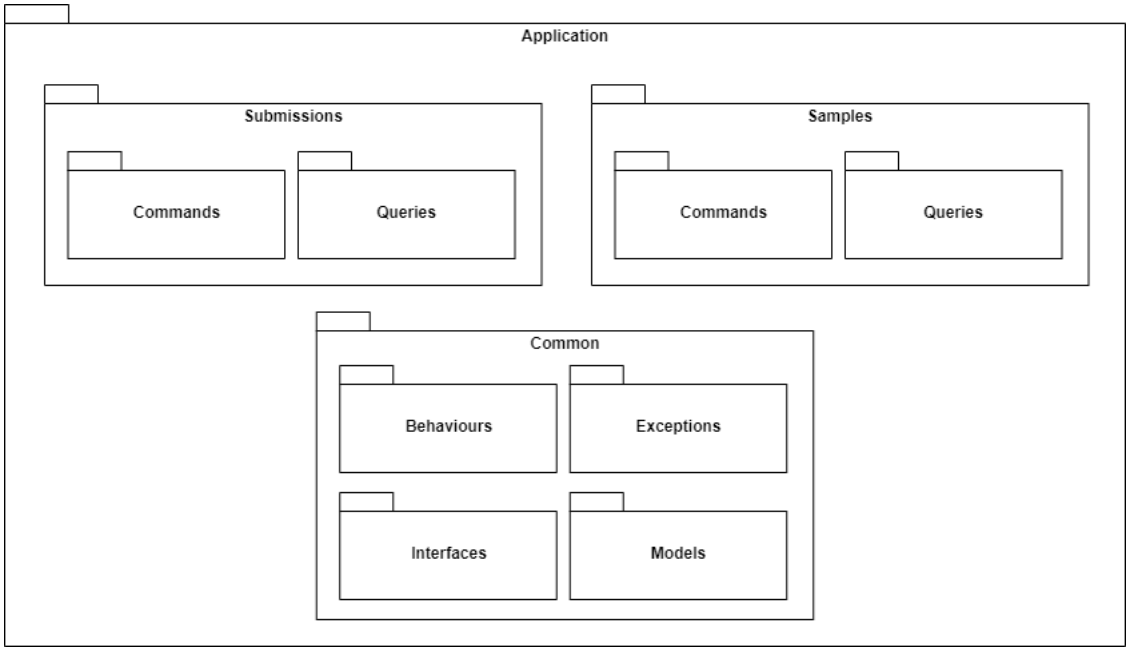


Figure 5.3: Application Module

5.1.3 Infrastructure Module

The Infrastructure module represents the infrastructure layer of the application, and this module implements all the interfaces of the Application module. It contains integration logic for used frameworks, external systems and data access. Figure 5.4 represents the module, along with the following submodules:

- **Cryptography**: contains the logic of the application related to cryptography. It is responsible for encrypting and decrypting data, responding to the non-functional requirement RNF04;
- **Emails**: contains the logic of the application related to emails. It is used for user sign-up and to recover user passwords;
- **Files**: contains the logic of the application related to file importation and exportation, responding to the functional requirements REQ-01, REQ-02, REQ-04 and REQ-06;
- **Identity**: contains the logic related to the Identity. Identity is a .Net API that manages user data for authentication and authorisation purposes, responding to the non-functional requirements RNF01 and RNF02;
- **Persistence**: contains the logic related the database access. It mainly consists of database objects and repositories since the application uses the Repository pattern, in detail in Section 1.2.

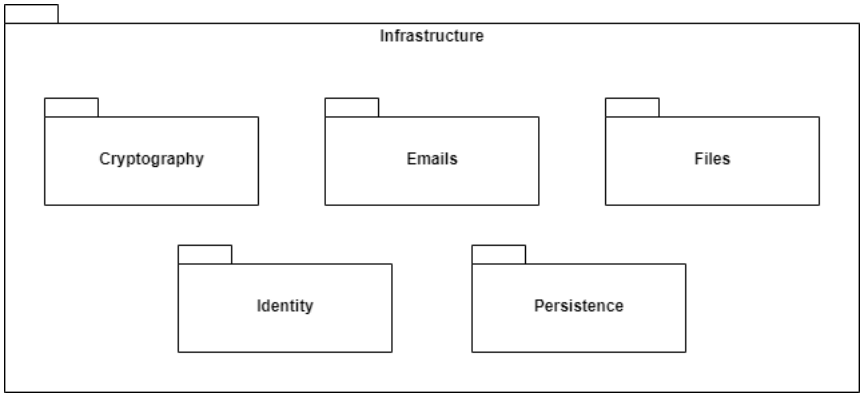


Figure 5.4: Infrastructure Module

5.1.4 Presentation Module

The Presentation module represents the presentation layer of the application. It contains the UI logic of the application and divides into the following modules, as displayed in Figure 5.5:

- **Controllers**: contains all the controllers of the application. It represents the controllers of the Model-View-Controller (MVC) pattern, in detail in Section 1.2;
- **Models**: contains all the view models of the application. It represents the view models of the MVC pattern;
- **Views**: contains all the views of the application. It represents the views of the MVC pattern.

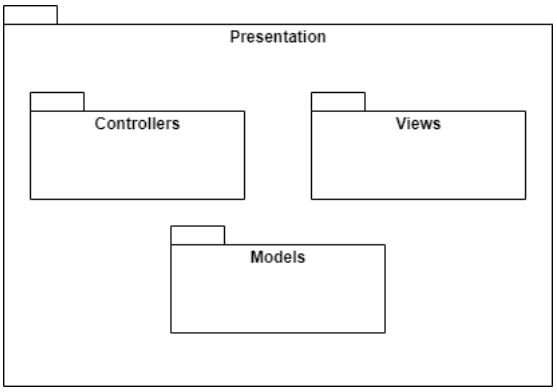


Figure 5.5: Presentation Module

5.2 Design Patterns

Design patterns are generic solutions to recurring problems in software design. Since they are tested and proven solutions, design patterns are considered best practices of software design. They improve the code maintainability and readability due to being well known by the community. However, when poorly applied, design patterns increase the application complexity and can deteriorate its performance. The following sections describe the adopted design patterns and show examples of their usage.

5.2.1 Command Query Responsibility Segregation

Command Query Responsibility Segregation (CQRS) is a design pattern that separates the model into a write model and a read model (Fowler 2011). The most fundamental idea comes from Command Query Separation (CQS), where an object's method should have two types, queries and commands. A query is a method that returns a value and does not change the state of the system. In contrast, a command changes the state of the system and does not return a value (Fowler 2005).

By separating the commands from queries, CQS increases the confidence in the use of queries and increases the awareness of the commands. CQRS takes this a step further. By separating the models into two, it is possible to scale reads and writes independently. With this design pattern, it is also possible to physically split the application and even the database. This separation can cause consistency problems, forcing the change to an event-based application, taking advantage of Event Sourcing.

CQRS taken to the extreme adds significant complexity to an application. This application uses CQRS by separating the reads and writes but keeping a single database and a shared model, reducing the added complexity of maintaining two different models.

Figure 5.6 displays the ImportTestsCommand and ImportTestsCommandHandler as an example of a command and command handler. And Figure 5.7 shows the GetSampleQuery and GetSampleQueryHandler as a contrast for a query and query handler. As the images show, the query handler returns a SampleDto while the command handler has no return value. Since the application also uses the mediator pattern (Section 5.2.2), it is only necessary to initialise the command/query, and the mediator will know which handler to use.

```
11 references | Bruno Rocha, 219 days ago | 1 author, 1 change
public class ImportTestsCommand : IRequest
{
    6 references | 3/3 passing | Bruno Rocha, 219 days ago | 1 author, 1 change
    public byte[] Content { get; set; }

    5 references | 3/3 passing | Bruno Rocha, 219 days ago | 1 author, 1 change
    public int HeaderRow { get; set; }

    6 references | 5/5 passing | Bruno Rocha, 219 days ago | 1 author, 1 change
    public string LaboratoryCodeColumn { get; set; }

    7 references | 5/5 passing | Bruno Rocha, 219 days ago | 1 author, 1 change
    public string ResultColumn { get; set; }
}

2 references | Bruno Rocha, 218 days ago | 1 author, 2 changes
public class ImportTestsCommandHandler : IRequestHandler<ImportTestsCommand>
{
    private readonly IXlsxFileReader fileReader;
    private readonly ISampleRepository sampleRepository;

    1 reference | Bruno Rocha, 218 days ago | 1 author, 2 changes
    public ImportTestsCommandHandler(IXlsxFileReader fileReader, ISampleRepository sampleRepository)
    {
        this.fileReader = fileReader;
        this.sampleRepository = sampleRepository;
    }

    53 references | 37/37 passing | Bruno Rocha, 218 days ago | 1 author, 2 changes
    public async Task<Unit> Handle(ImportTestsCommand request, CancellationToken cancellationToken)
    {
        foreach (var keyValuePair in this.fileReader.ReadSamplesTestsFile(request))
        {
            var sample = await this.sampleRepository.GetSampleByLabCodeAsync(keyValuePair.Key);
            sample.AddTest(keyValuePair.Value);
            await this.sampleRepository.UpdateAsync(sample);
        }

        return Unit.Value;
    }
}
```

Figure 5.6: Command and command handler example

```
5 references | Bruno Rocha, 151 days ago | 1 author, 1 change
public class GetSampleQuery : IRequest<SampleDto>
{
    7 references | 4/4 passing | Bruno Rocha, 151 days ago | 1 author, 1 change
    public int Id { get; set; }
}

2 references | Bruno Rocha, 151 days ago | 1 author, 1 change
public class GetSampleQueryHandler : IRequestHandler<GetSampleQuery, SampleDto>
{
    private readonly ISampleRepository sampleRepository;

    1 reference | Bruno Rocha, 151 days ago | 1 author, 1 change
    public GetSampleQueryHandler(ISampleRepository sampleRepository)
    {
        this.sampleRepository = sampleRepository;
    }

    53 references | 37/37 passing | Bruno Rocha, 151 days ago | 1 author, 1 change
    public async Task<SampleDto> Handle(GetSampleQuery request, CancellationToken cancellationToken)
    {
        var sample = await this.sampleRepository.GetAsync(request.Id);
        if (sample == null)
        {
            throw new ValidationException("Id", "Sample doesn't exist.");
        }

        return new SampleDto
        {
            Id = sample.Id,
            PatientNumber = sample.PatientNumber,
            CovidCode = sample.CovidCode.Value,
            LaboratoryCode = sample.LaboratoryCode.FriendlyValue,
            CollectedAt = sample.CollectedAt,
            CreatedAt = sample.CreatedAt
        };
    }
}
```

Figure 5.7: Query and query handler example

5.2.2 Mediator

A Mediator serves as an abstraction between different objects. Instead of an object communicating with the other directly, it only needs to communicate with the Mediator, providing loose coupling. It simplifies the way objects work since they only need to know the Mediator instead of being directly coupled with many other objects (Gamma et al. 1994).

The Mediator centralises the control. Instead of implementing new behaviour across multiple objects, it is possible to implement those behaviours in the Mediator.

```
var sample = await this.mediator.Send(new GetSampleQuery
{
    Id = id
});
```

Figure 5.8: Mediator example

Figure 5.8 is an example of the Mediator pattern applied in the application. In the example, the Mediator abstracts the query from the responsible handler. It also abstracts a collection of behaviours that execute before the handler in the form of a Pipeline (Section 5.2.3).

5.2.3 Pipeline

The Pipes and Filters or Pipeline pattern decomposes a complex process into simpler tasks that can be reused. Each task, or filter, needs to follow a standardised data format and together, they combine into a pipeline. This pattern improves the code reusability, performance and scalability since each filter can be deployed and scaled independently (Microsoft 2017).

With the Mediator pattern, it is possible to add a pipeline to every request made to the Mediator. Instead of adding the pipeline or a complex process to each object, the Mediator can run the pipeline for them, further improving the loosed coupling and code reusability.

```
2 references | Bruno Rocha, 45 days ago | 1 author, 2 changes
public class ValidationBehaviour<TRequest, TResponse> : IPipelineBehavior<TRequest, TResponse>
    where TRequest : IRequest<TResponse>
{
    private readonly IEnumerable<IValidator<TRequest>> validators;

    2 references | 2/2 passing | Bruno Rocha, 45 days ago | 1 author, 2 changes
    public ValidationBehaviour(IEnumerable<IValidator<TRequest>> validators)
    {
        this.validators = validators ?? new List<IValidator<TRequest>>();
    }

    2 references | 2/2 passing | Bruno Rocha, 290 days ago | 1 author, 1 change
    public async Task<TResponse> Handle(TRequest request, CancellationToken cancellationToken, RequestHandlerDelegate<TResponse> next)
    {
        if (this.validators.Any())
        {
            var context = new ValidationContext<TRequest>(request);

            var validationResults = await Task.WhenAll(this.validators.Select(v => v.ValidateAsync(context, cancellationToken)));
            var failures = validationResults.SelectMany(r => r.Errors).Where(f => f != null).ToList();

            if (failures.Count != 0)
                throw new ValidationException(failures);
        }
        return await next();
    }
}
```

Figure 5.9: Filter example

Figure 5.9 shows an example of a filter used in the application, the ValidationBehaviour. This class is responsible for validating the command and queries before they reach the handler.

```
services.AddValidatorsFromAssembly(Assembly.GetExecutingAssembly());
services.AddMediatR(Assembly.GetExecutingAssembly());
services.AddTransient(typeof(IPipelineBehavior<,>), typeof(ValidationBehaviour<,>));
```

Figure 5.10: Pipes and Filters with Mediator

```
2 references | Bruno Rocha, 219 days ago | 1 author, 1 change
public class ImportTestsCommandValidator : AbstractValidator<ImportTestsCommand>
{
    1 reference | Bruno Rocha, 219 days ago | 1 author, 1 change
    public ImportTestsCommandValidator()
    {
        RuleFor(c => c.Content)
            .NotNull()
            .NotEmpty();
        RuleFor(c => c.HeaderRow)
            .NotNull()
            .GreaterThanOrEqualTo(1);
        RuleFor(c => c.LaboratoryCodeColumn)
            .MinimumLength(1)
            .Matches(@"^[a-zA-Z]{1}$")
            .NotNull();
        RuleFor(c => c.ResultColumn)
            .NotNull()
            .MinimumLength(1)
            .Matches(@"^[a-zA-Z]{1}$");
    }
}
```

Figure 5.11: Validator example

This class only needs to be registered on the Mediator, and the Mediator is responsible for running it before any command or query (Figure 5.10). Each command and query needs to have a validator defined to use the ValidationBehaviour. The validator only needs to have the validation rules leaving the error handling to the ValidationBehaviour class (Figure 5.11).

5.2.4 Repository

According to Fowler et al. (2002) the Repository pattern serves as an abstraction between the domain and the data access layer. It acts as an in-memory domain object collection, where objects can be added, updated or deleted from the Repository, and the Repository is responsible for the proper database operations. This pattern provides an object-oriented view of the persistence layer, creating a clear separation between the domain and the data access layer.

Using the Repository pattern further improves the code testability since it is possible to mock the Repository. Because it is an abstraction of the data access layer, it is easy to replace the used object-relational mapping providing a loose coupling between the domain and the data access layer.

In Figure 5.12, it is possible to see the SampleRepository implemented in the application. Since the Repository only receives and returns domain objects, the CreateSampleCommand can treat the Repository as a collection and call the AddAsync method directly (Figure 5.13).

```
6 references | 3/3 passing | Bruno Rocha, 152 days ago | 1 author, 3 changes
public async Task<Domain.Entities.Sample> AddAsync(Domain.Entities.Sample entity)
{
    const string query = "INSERT INTO Samples " +
        "(PatientNumber, CovidCode, LaboratoryCode, LaboratoryCodeLetter, LaboratoryCodeNumber, " +
        "CreatedDateStamp, CollectedAt, CreatedAt) " +
        "OUTPUT INSERTED.ID " +
        "VALUES (@PatientNumber, @CovidCode, @LaboratoryCode, @LaboratoryCodeLetter, " +
        "@LaboratoryCodeNumber, @CreatedDateStamp, @CollectedAt, @CreatedAt);";

    var dbo = this.sampleAdapter.Adapt(entity);

    var parameters = new DynamicParameters(dbo);

    var sampleId = await this.dapperSql.ExecuteScalarAsync<int>(query, parameters);

    dbo.Id = sampleId;

    return this.sampleAdapter.Adapt(dbo);
}
```

Figure 5.12: Repository example

```
53 references | 37/37 passing | Bruno Rocha, 151 days ago | 1 author, 3 changes
public async Task<string> Handle(CreateSampleCommand request, CancellationToken cancellationToken)
{
    var sample = new Sample(
        request.PatientNumber,
        CovidCode.Parse(request.CovidCode),
        request.CollectedAt);

    var labCodeNumber = await this.sampleRepository.GetLastLabCodeNumberOfDayAsync(sample.CreatedDateStamp);
    sample.GenerateLabCode(labCodeNumber + 1);
    await this.sampleRepository.AddAsync(sample);

    return sample.LaboratoryCode.FriendlyValue;
}
```

Figure 5.13: Example from a call to a repository

5.2.5 Model-View-Controller

According to Microsoft (2020c), the Model-View-Controller (MVC) is a design pattern that "separates an application into three main groups of components: Models, Views, and Controllers". Each request made to the application routes to a Controller, which sends commands or queries to the model. The Controller is also responsible for the View to return and display to the user, feeding the View with the Model data required.

In this pattern, the model represents the business logic, and the View represents the user interface content. Both the View and the Controller depend on the model; however, the model depends on neither. The separation between the model and the user interface logic allows the model to be built and tested independently. Since the user interface changes more frequently than the model, this separation is a key benefit of the MVC pattern.

An application can use ViewModels to further extend the separation between the View and the model. Instead of having the View depend on the model, the View can depend on a ViewModel, which represents the data displayed on the View. In this case, the Controller is responsible for populating the ViewModel data with the domain data.

Figure 5.14 shows the SampleController's action GetSampleWithPagination. The Controller makes a query to the Mediator and uses the data returned to populate the GetSamplesWithPaginationViewModel. Then, it returns the correspondent View with the GetSamplesWithPaginationViewModel.

```
[HttpGet]
[Route("Samples")]
4 references | Bruno Rocha, 187 days ago | 1 author, 1 change
public async Task<IActionResult> GetSamplesWithPagination(string searchString, int? pageIndex, int? pageSize,
    string startDate, string endDate, int[] selectedColumns = null)
{
    try
    {
        var paginatedList = await this.mediator.Send(new GetSamplesWithPaginationQuery
        {
            PageIndex = pageIndex ?? 0,
            SearchString = searchString,
            PageSize = pageSize ?? 10,
            StartDate = startDate.ToDateTime(),
            EndDate = endDate.ToDateTime()
        });

        return View(new GetSamplesWithPaginationViewModel(selectedColumns?.ToList())
        {
            PaginatedList = paginatedList,
            PageSize = pageSize ?? paginatedList.PageSize,
            SearchString = searchString,
            StartDate = startDate.ToDateTime()?.ToString("s"),
            EndDate = endDate.ToDateTime()?.ToString("s"),
        });
    }
    catch (Exception e)
    {
        this.logger.LogError(e.Message);

        return RedirectToAction("Error", "Home");
    }
}
```

Figure 5.14: Example from a Controller

5.3 Cryptography

RNF04 states that the application must encrypt the personal data stored. The cryptography module situated in the infrastructure layer addresses this non-functional requirement, and it contains encryption and decryption methods using the Advanced Encryption Standard algorithm, also known as Rijndael (Daemen and Rijmen 1999).

The National Institute of Standards and Technology established the AES as the standard encryption algorithm. It is a symmetrical block cypher algorithm "capable of using cryptographic keys of 128, 192 and 256 bits to encrypt and decrypt data in blocks of 128 bits" (Dworkin et al. 2001). Since this is a symmetric-key algorithm, the key used to encrypt and decrypt the data must be the same. Only using this key is not enough because if the same data is encrypted twice, it will result in the same cyphertext. The initialisation vector solves this issue since it is a unique value generated for every new message that is being encrypted. The IV is usually stored together with the encrypted data.

Figure 5.15 shows how the application encrypts the data. The key used is the same for each encryption, and the application concatenates the IV with the data. As for the decryption, the application gets the IV from the encrypted data. Then, it uses the IV and the key to decrypt the rest of the encrypted data (Figure 5.16).

```
6 references | 5/5 passing | Bruno Rocha, 243 days ago | 1 author, 1 change
public byte[] Encrypt(string data)
{
    if (string.IsNullOrWhiteSpace(data))
    {
        throw new ArgumentNullException("Data to encrypt must be defined");
    }

    using (var aes = Aes.Create())
    {
        aes.Key = this.key;

        var encryptor = aes.CreateEncryptor(aes.Key, aes.IV);

        using (var memoryStream = new MemoryStream())
        {
            using (var cryptoStream = new CryptoStream(memoryStream, encryptor, CryptoStreamMode.Write))
            {
                cryptoStream.Write(aes.IV);
                using (var streamWriter = new StreamWriter(cryptoStream))
                {
                    streamWriter.Write(data);
                }
                return memoryStream.ToArray();
            }
        }
    }
}
```

Figure 5.15: Data Encryption

```
5 references | 3/3 passing | Bruno Rocha, 162 days ago | 1 author, 2 changes
public string Decrypt(byte[] data)
{
    if (data == null || data.Length <= 0)
    {
        throw new ArgumentNullException("Data to decrypt must be defined");
    }

    using (var aes = Aes.Create())
    {
        aes.Key = this.key;
        aes.IV = data.Take(IVByteSize).ToArray();

        var decryptor = aes.CreateDecryptor(aes.Key, aes.IV);

        using (var memoryStream = new MemoryStream(data.Skip(IVByteSize).ToArray()))
        {
            using (var cryptoStream = new CryptoStream(memoryStream, decryptor, CryptoStreamMode.Read))
            {
                using (var streamReader = new StreamReader(cryptoStream))
                {
                    return streamReader.ReadToEnd();
                }
            }
        }
    }
}
```

Figure 5.16: Data Decryption

5.4 Functional Requirements

This section describes REQ-01 and REQ-05 from a system perspective. It presents, for each requirement, a sequence diagram and specifies how each part of the system interacts. This section further compliments the rest of the chapter, giving a detailed perspective of the system implementation.

5.4.1 REQ-01: Import samples from health centres

REQ-01: Import samples from health centres is an essential functional requirement for the P.Porto laboratory. It enables the actor to import samples from several sources, even if each

source has a different format.

The Laboratory Administrator initialises this requirement by sending an HTTP POST request to the system. The request contains the file content and the columns responsible for the COVID code, collection date and patient number. The SamplesController receives the request and verifies if the authenticated user is authorised to make this request. Then, the SamplesController creates the command to import the samples and sends it to the Mediator. Before sending the command to the respective handler, the Mediator runs the pipeline of behaviours configured. In this pipeline, there is only a single behaviour, the ValidationBehaviour. This behaviour asks the respective Validator to validate the command, throwing an exception if the Validator returns any error.

The command handler sends the command information to the file reader, whose job is to read the file content, translate it into domain objects and return them. In this case, the file reader returns samples to the handler. Then, for each sample, the handler gets the last laboratory code number of the day from the SampleRepository, generates the following laboratory code (Section 4.1.2 explains the laboratory code concept) and adds the sample to the SampleRepository.

Figure 5.17 displays the sequence diagram for this functional requirement.

5.4.2 REQ-05: Search samples

REQ-05: Search samples is another crucial functional requirement for the application. This requirement allows the actor to search the samples using multiple filters. The actor can select a temporal interval and can search by text. The search by text accepts COVID codes and Laboratory Codes. Since the organisation identifier composes the COVID code, it is possible to search by organisation as well. The system response can contain several samples, and because of that, the system returns a paginated response.

The Laboratory Administrator, or the Sample Collector, initialises this requirement by sending an HTTP GET request to the system (Figure 5.18). The request sends the search text, the page index of the paginated table, the number of items for each page and the start and end date to filter the samples. The SamplesController receives the request and verifies if the user is authorised to make this request. Then, the SamplesController creates the query and sends it to the Mediator. The Mediator runs the pipeline of behaviours, similar to the REQ-01 described previously.

After the ValidationBehaviour step, the Mediator sends the query to the respective handler. The handler asks the SampleRepository for the total count of samples that exist and match the filters passed. Then, the handler gets the samples from the SampleRepository. Here, the handler passes, to the SampleRepository, the page index and the size of each page so that the SampleRepository only gets the necessary amount of data from the database. The handler creates the paginated list to return. Since the handler will return this paginated list to the Presentation layer, it uses the SampleDto instead of the Domain object. The SampleDto contains information about the sample and the latest test result. As such, the handler gets the latest of each sample from the TestRepository. Finally, the handler returns the paginated list to the SamplesController, which passes the same list to the View and returns the View to the user.

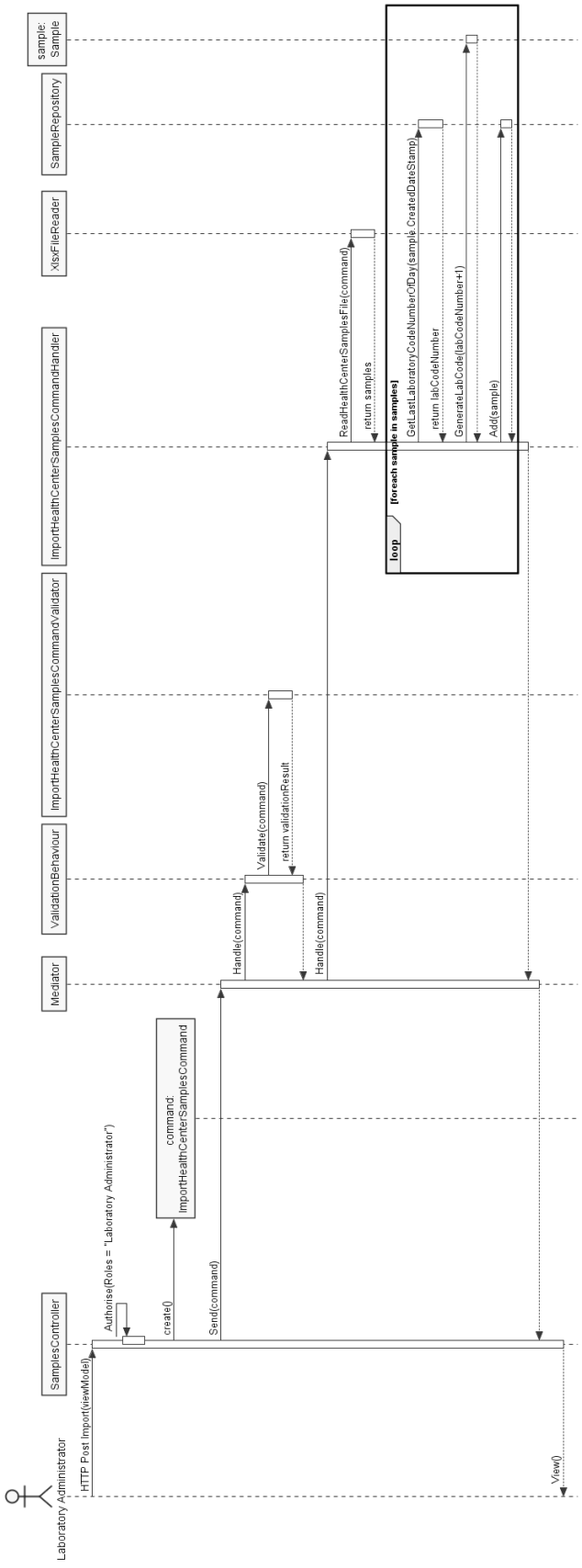


Figure 5.17: REQ-01: Sequence Diagram

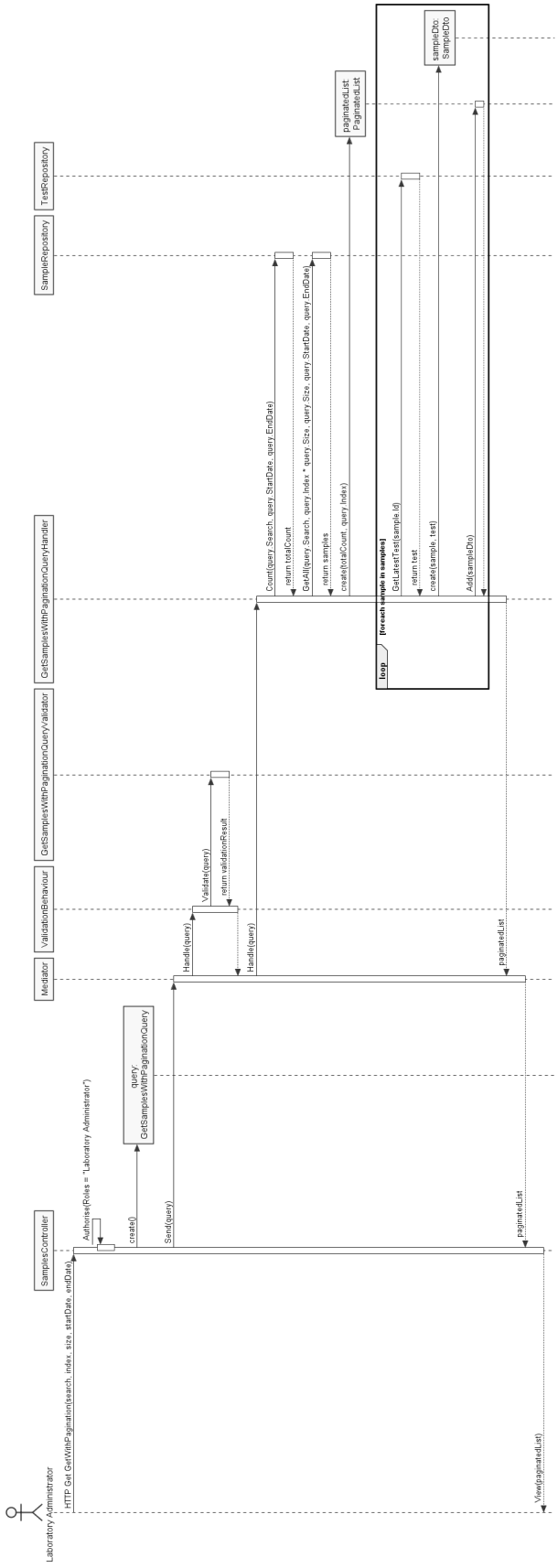


Figure 5.18: REQ-05: Sequence Diagram

5.5 Tests

Software testing is a crucial part of software development since it ensures that the software meets the defined requirements and helps to identify possible flaws. This section explains the different types of software testing done and their value to this system's development.

5.5.1 Unit tests

A unit test is an automated test that verifies a small piece of code, in other words, a unit. Most authors agree with this definition; however, they diverge in whether the unit tests should run in an isolated manner. The *mockist* approach isolates the System Under Test (SUT) from its dependencies, using test doubles (add a note with definition). In contrast, the classical approach only isolates the unit tests from each other and utilises the dependencies of the SUT in the unit tests, with a few exceptions (Khorikov 2020).

This application's unit tests follow the *mockist* approach since it provides better granularity and, in case of a test failure, it is easier to detect which functionality has failed. The unit tests rely on mocks to replace the SUT's dependencies and their interactions. The mock framework used is the Moq, and the unit tests framework is the Xunit. The AutoFixture framework is also used to create random values for the unit tests.

The unit tests follow the Arrange, Act, Assert (AAA) pattern:

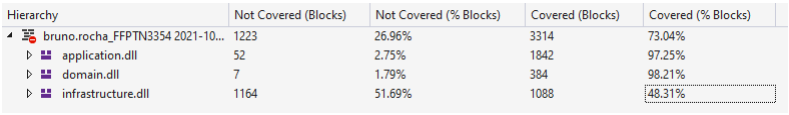
- The Arrange section of the unit test is where the objects and values used are initialised;
- The Act section invokes the SUT;
- The Assert verifies that the SUT behaves as expected (Microsoft 2021).

Figure 5.19 shows a unit test to the GetSampleQueryHandler as an example. This test intends to validate the behaviour of the GetSampleQueryHandler if the sample to return does not exist, and it utilises a mock object to replace the SampleRepository implementation. In the Arrange section, the unit test initialises the GetSampleQuery, the GetSampleQueryHandler, which is the SUT, and it setups the mock behaviour to return a null Sample on GetAsync. The Act section invokes the SUT, and the Assert section verifies that the expected behaviour is to throw a ValidationException. During the Assert, the unit test also validates if the set up mock behaviour was invoked.

```
[Fact]
0 references | Bruno Rocha, 39 days ago | 1 author, 1 change
public async Task Handle_SampleDoesntExist()
{
    //Arrange
    var query = this.fixture.Create<GetSampleQuery>();

    this.sampleRepositoryMock.Setup(x => x.GetAsync(query.Id))
        .ReturnsAsync(default(Sample));

    var sut = this.CreateGetSampleQueryHandler();

    //Act
    Func<Task> action = async () => await sut.Handle(query, CancellationToken.None);

    //Assert
    await action.Should().ThrowAsync<ValidationException>();

    this.sampleRepositoryMock.VerifyAll();
}
```

Figure 5.19: GetSampleQueryHandler Unit Test

The unit tests cover all the application layers except the Presentation layer due to only having the user interface logic. The code coverage of the unit tests reached 73.04% (Figure 5.20).



Hierarchy	Not Covered (Blocks)	Not Covered (% Blocks)	Covered (Blocks)	Covered (% Blocks)
bruno.rocha_FFPTN3354 2021-10...	1223	26.96%	3314	73.04%
application.dll	52	2.75%	1842	97.25%
domain.dll	7	1.79%	384	98.21%
infrastructure.dll	1164	51.69%	1088	48.31%

Figure 5.20: Code Coverage of Unit Tests

5.5.2 User Acceptance tests

User acceptance tests (HISs) primary objective is to ensure that a system meets the business requirements. The end-user usually does these tests and determines if the system satisfies the acceptance criteria. UATs are relevant tests for a new information system since they help identify missing requirements or miss interpretation of the existent requirements (Hambling and Goethem 2013).

The development of this system followed a scrum methodology. This methodology relies on incremental development where each iteration, also known as a sprint, has a predefined duration of time. Each sprint consists of a planing, to define what is being done, a development phase and a delivery. In the delivery phase of the sprint, the end-users would test the system's new functionalities and either accept the functionality or ask for changes. This step identified flaws and missing requirements that led to a final functional requirement list quite different from the initial one (Section 4.1.3).

Each functional requirement directly translates to a UAT. Table 5.1 shows the functional requirements and if they were accepted or not by the end of the development.

Table 5.1: Accepted Functional Requirements

Identifier	Description	Accepted
REQ-01	Import Samples from health centres	Yes
REQ-02	Import P.Porto submissions	Yes
REQ-03	Register a new sample	Yes
REQ-04	Import COVID-19 test results	Yes
REQ-05	Search samples	Yes
REQ-06	Export samples	Yes
REQ-07	Update sample	Yes
REQ-08	Delete sample	Yes
REQ-09	Get sample details	Yes
REQ-10	Register a new P.Porto submission	Yes
REQ-11	Update P.Porto submission	Yes
REQ-12	Delete P.Porto submission	Yes
REQ-13	Associate sample to P.Porto submission	Yes
REQ-14	Decouple sample from P.Porto submission	Yes
REQ-15	Search submissions	Yes
REQ-16	Get submission details	Yes

Chapter 6

Evaluation

This chapter's objective is to evaluate the developed system. The P.Porto laboratory answered a questionnaire to determine the perceived usefulness and ease of use. Afterwards, the system is compared with what would an ideal LIS be like from a functional point of view.

6.1 Perceived Usefulness and Perceived Ease of Use

Davis (1989) published an article validating perceived usefulness and ease of use as possible determinants of computer usage. According to him, perceived usefulness is "the degree to which a person believes that using a particular system would enhance his or her job performance.". On the other hand, perceived ease of use is "the degree to which a person believes that using a particular system would be free of effort.".

During his research, Fred Davids developed perceived usefulness and ease of use measures and validated them in two empirical studies. He generated initial scale items and pretested them in a pilot study. Then, he refined the scale items, tested them on the first study, and repeated the process for the second study. Perceived usefulness correlated .63 with self-reported system use for the first study and .85 for the second, while perceived ease of use correlated .45 and .69, correspondingly. These numbers show that both perceived usefulness and ease of use correlated significantly with self-reported use, but perceived usefulness had a stronger link. Fred Davids solidifies this statement by examining the combined direct effect of the two variables in a regression analysis. The regression results confirmed the statement and suggested that ease of use could be an antecedent to usefulness, which, in turn, would lead to usage, causing a chain of causality.

Section 3.1.1 identifies the opportunity to build a new information system for PORTIC. That opportunity arose from the administrative burden suffered by the laboratory. They deal with a significant amount of data and spend most of their time managing that information. Using the scale measures for perceived usefulness and ease of use makes it possible to evaluate the user acceptance and correlate to possible application usage. For this evaluation, a questionnaire with the scale measures was made to the laboratory. The questionnaire has two sections with six questions each, one for perceived usefulness and another for ease of use. The answers go from a scale of one to seven, where one is extremely unlikely, and seven is extremely likely.

Figure 6.1 displays the laboratory answers for the perceived usefulness. They answered six for all questions but one, where they answered seven. Given that perceived usefulness highly correlates to self-reported use, it is possible to conclude that the system would be extremely useful and frequently used by the laboratory.

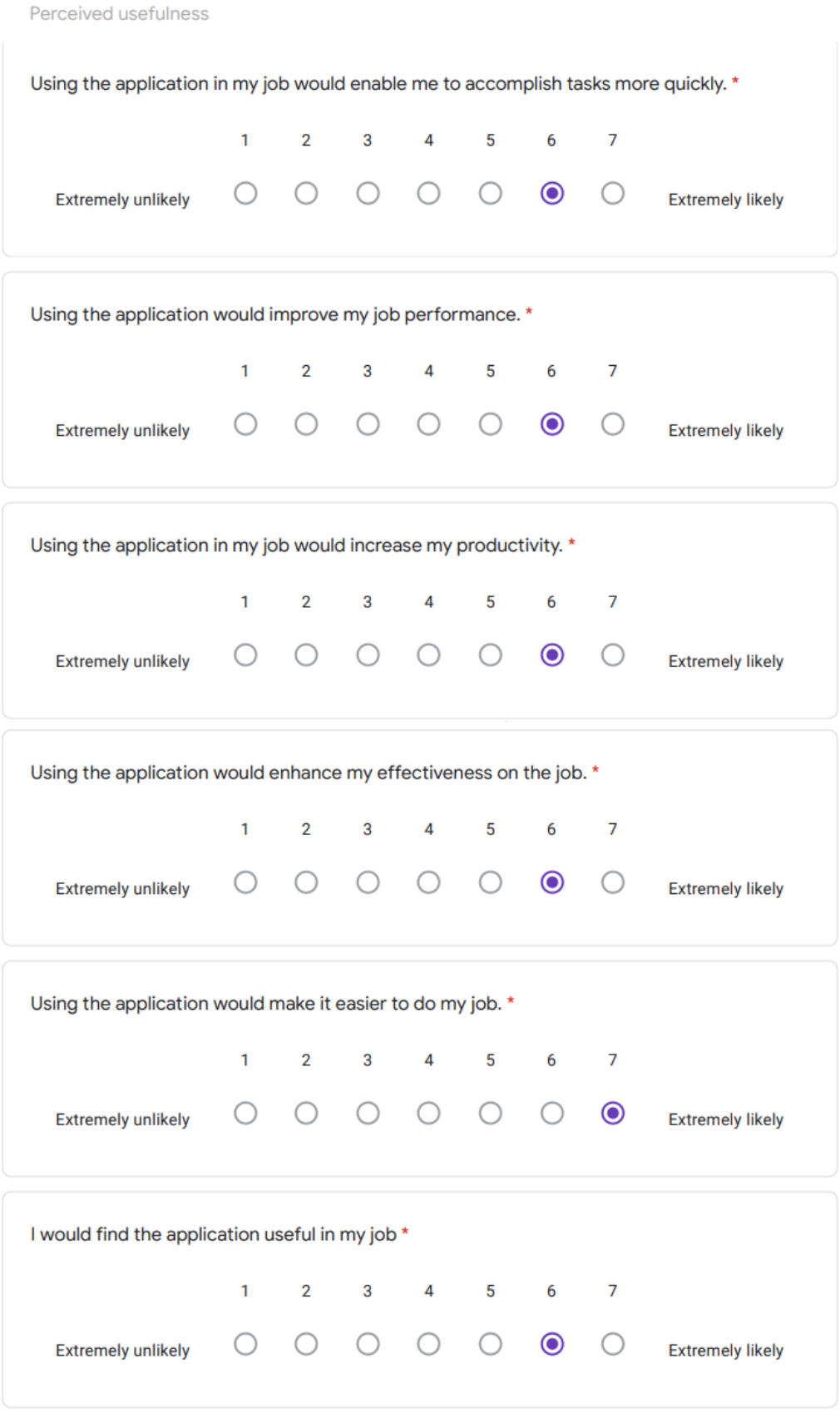


Figure 6.1: Perceived Usefulness Questionnaire

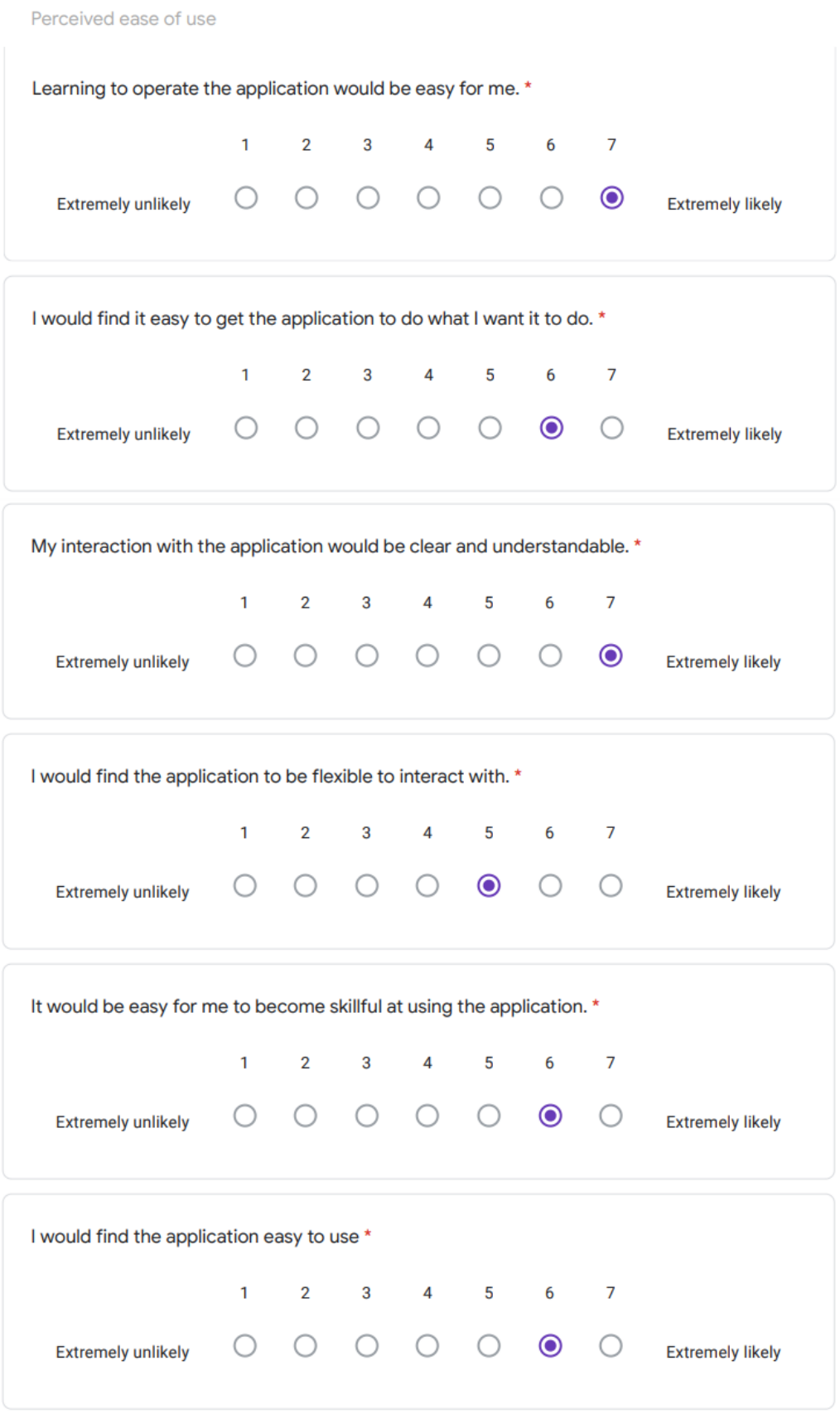


Figure 6.2: Perceived Ease of Use Questionnaire

As for the perceived ease of use, the laboratory answers were spread between fives and sevens (Figure 6.2). This result is very similar to the perceived usefulness, and it confirms the previous conclusion.

The laboratory answered this questionnaire after all the functional requirements were delivered and accepted by the user acceptance tests. The laboratory also had a last session to ask questions and solve doubts about the system before answering the questionnaire.

6.2 The Ideal Laboratory Information System

Section 2.3 presents the ideal LIS from a functional point of view, according to Sepulveda and D. S. Young (2013). This section compares the system developed with the ideal LIS. Table 6.1 presents the functionalities that an ideal LIS should have and the state of those functionalities in the developed system.

Table 6.1: Ideal LIS Comparison

Functionality	**State**
Information security	Incomplete
Test ordering	Completed
Specimen collection, accessioning and processing	Completed
Analytic phase	Not implemented
Result entry and validation	Incomplete
Result reporting	Incomplete
Notification management	Not implemented
Data mining and cross-sectional reports	Not implemented
Method validation	Not implemented
Quality management	Not implemented
Administrative and financial issues	Not implemented

Test ordering and specimen collection, accessioning and processing is fully implemented and follows the ideal LIS description. On the other hand, there are some functionalities that incomplete. Information security is present but lacking functionalities, namely the lack of user roles. Initially, the system required multiple user roles (Section 4.1.1); however, the requirements changed over time, and due to time constraints, only one user role was configured. Even though there is only one user role, the system is prepared to have more.

Result entry and validation is another incomplete functionality of an ideal LIS. The system allows the user to create, edit and delete samples and P.Porto submissions data; however, it does not have a step to validate the data upon file importing. This functionality was not a part of the system requirements, but it would improve its ease of use.

The system supports result reporting through REQ-06: Export Samples. The downside is that the user still needs to submit those reports to or directly to the person. For this functionality to be completed, the system would have to send the reports for the user. This was considered as a functional requirement, but it was discarded due to time constraints.

As for the not implemented functionalities:

- **Analytic phase**: was not implemented because it was not a part of the functional requirements;
- **Notification management**: was not implemented because it was not a part of the functional requirements. However, notification management would improve the usefulness of the system and increase the overall performance of the laboratory;
- **Data mining and cross-sectional reports**: was not implemented because it was not a part of the functional requirements. Now that the data is normalised and in the system's database, there is an opportunity to explore that data for research on COVID-19;
- **Method validation**: was not implemented because it does not fit the system's purpose; the laboratory does not make clinical assays;
- **Quality management**: was not implemented because it was not a part of the functional requirements;
- **Administrative and financial issues**: the financial issues support was not implemented because it was not a part of the functional requirements. As for the administrative issues, initially, it was considered a functional requirement to manage user permissions; however, due to time constraints, this was descoped, and the system would only function with one user role.

The system still has a long way to go to be considered an ideal LIS. There are incomplete functionalities that would improve the system's usefulness and ease of use. Some of the not implemented functionalities, like the data mining and cross-sectional reports, represent missed opportunities. Nevertheless, the system responds to most of the laboratory requirements, and the missing functionalities of an ideal LIS are justified by time constraints. The laboratory was under pressure due to the high amount of testing for COVID-19, and the information management was the biggest bottleneck. The system was necessary to solve this bottleneck quickly. As such, the system's functionalities represent the essential to manage the information and mitigate the bottleneck.

Chapter 7

Conclusions

This chapter summarises each chapter of this dissertation and what they concluded. Discusses the objectives accomplished and not accomplished, and it also describes the limitations and future work of this dissertation.

7.1 Summary

This dissertation's main objective is a LIS development that allows the PORTIC laboratory to manage all the information of the COVID-19 tests. This includes the entire sample flow, from the test application, passing through sample collection and analysis, to the diagnose and test reporting.

Chapter 1 contextualises this dissertation and defines the problem, objectives and expected results. Afterwards, chapter 2 analyses the current state of health information systems, LIS and medical informatics. This chapter also analyses market solutions for the problem, such as the Belgian LIMS for COVID-19 or the CrelioHealth. It was possible to conclude that most of these LIS satisfy essential features for the laboratories, but only a small portion of them have an answer for particular COVID-19 requirements. Furthermore, the PORTIC laboratory needs specific requirements, such as the importation of custom forms and the generation of custom laboratory codes, that none of the solutions analysed satisfy.

Chapter 3 analyses the value of the current dissertation. By applying the NCD model, it was possible to understand and analyse the opportunity of this dissertation. This value analysis concludes that there is a lack of information systems that support COVID-19 testing. The value proposition canvas demonstrated how the system would relieve the pains and create gains for the users. And the QFD translated the customer needs into technical requirements for the software.

Chapter 4 describes the functional and non-functional requirements for the system. It explains the process of requirements elicitation and details the domain model and the system actors. This entire process of requirements elicitation and requirements specification is crucial to the architectural design and technology definition. This chapter analysed relevant technologies, concluding that the best technology to use is ASP.NET Core. It also analysed possible architectural styles to adopt, concluding that clean architecture was the best option. This chapter also defined the deployment view based on the single server that the P.Porto provided. Lastly, the database model was also determined, taking into consideration the functional and non-functional requirements.

Chapter 5 explains how the system was implemented. It describes the code structure, detailing each module and relating them to the layers of the clean architecture. It explains the

adopted design patterns, the added value of using them and demonstrates evidence from the code. This chapter also describes how the non-functional requirement of data encryption was tackled. It analyses two functional requirements and how they were implemented, utilising sequence diagrams to illustrate the implementation. Finally, it gives an overview of the unit tests and how they were implemented while explaining how the user acceptance tests were done.

Chapter 6 evaluates the developed system by determining the perceived usefulness and ease of use through a questionnaire. This allowed to conclude that the system would be extremely useful and easy to use, with some space for improvements. This chapter also compares the system with an ideal LIS from a functional point of view. It concludes that the system still missing some functionalities of an ideal LIS but accomplishes its primary objective of mitigating the bottleneck of the laboratory.

7.2 Objectives Accomplished

Table 7.1 presents the objectives proposed in Section 1.3 and their state of accomplishment.

Table 7.1: Objectives Accomplished

Objectives	**State**
Requirements elicitation and specification	Accomplished
Investigation and analysis of existing solutions to similar problems	Accomplished
Investigation and analysis of different architectural designs	Accomplished
Identify and justify the most viable alternative	Accomplished
Data importation from different external sources	Accomplished
Importation of RT-PCR test results	Accomplished
Report generation	Accomplished
Authentication and authorisation	Partially Accomplished
Management of users authorisations	Not Accomplished
Unit tests	Accomplished
Integration tests	Not Accomplished
Acceptance tests	Accomplished

The system did not accomplish two of the main objectives proposed, management of users authorisations and integration tests. They are both justified by requirements change, time constraints and urgency. During the development, the laboratory identified new functional requirements that surpassed these objectives on priority. Given the increasing amount of COVID-19 testing, the laboratory needed the system as soon as possible. As such, this objective was descoped. As for the integration tests, they are an essential part of software testing. However, for the same reasons as previously explained, the objective was not accomplished.

Authentication and authorisation is only objective that is partially accomplished. The system supports authentication and authorisation; however, the objective comprised multiple user

roles, and the system only supports one user role, the Laboratory Administrator. Nevertheless, the system is prepared to accommodate new user roles with a few quick changes to the code.

The rest of the objectives were fully accomplished.

7.3 Limitations and Future Work

Even though the developed system accomplished most objectives, there are still objectives to complete and opportunities for improvement. The system is still missing more user roles to match the actors defined in Section 4.1.1. The Laboratory Administrators cannot manage user authorisations on the system, but they will need this functionality with the addition of new user roles. Integration tests are also missing, and they are essential to ensure the system works as expected in an automated way.

Section 6.2 compares the developed system with the ideal LIS from a functional point of view. That comparison uncovered multiple opportunities to improve the system. Automated reporting would remove manual work and improve the laboratory performance, as well as notification management. Data mining and cross-sectional reports is a significant opportunity that this system brought since it collects data about the tests and patient symptoms. By exploring this data, it would be possible to use this system for COVID-19 research.

7.4 Final Assessment

This dissertation's main objective was to develop a LIS for the PORTIC laboratory to manage the information on COVID-19 testing. That meant that the system would have to support data importation, report generation and exportation and manage the entire sample flow.

Chapter 2 showed that information systems and computer applications have been applied to healthcare for a long time, given the potential to improve medical processes. Information systems play an important part in medical informatics since they can collect, process, and store data. They can reduce costs, improve the outcomes, and enable new organisational structures, such as virtual organisations where employees can work from anywhere. Information systems can also be a great tool when dealing with outbreaks of infectious diseases, such as COVID-19. One of the biggest challenges in these outbreaks is controlling the spread, and time is crucial for it. Chapter 2 then explained that an ideal LIS should have functionalities such as information security, test ordering, specimen collection, accessioning and processing, result entry, validation and reporting, among others. Finally, it compared possible market solutions that corresponded to the required functionalities of PORTIC's laboratory. Most existing solutions did not have essential functionalities for COVID-19 testing. Only one of them had the necessary functionalities, which was the Belgian LIMS. This system was designed and developed specifically to deal with COVID-19 testing. However, the system is not open-source, and it is not available on the market.

To conclude, the developed system answers PORTIC's laboratory needs, but it can also be reused by other laboratories that do COVID-19 testing. Therefore, the system is an innovation on COVID-19 testing since it tackles a new reality that generated new constraints to laboratories.

References

Arokiasamy, J. et al. (Feb. 2000). "Recommendations of the International Medical Informatics Association (IMIA) on education in health and medical informatics". In: *Methods of Information in Medicine* 39 (3). (Accessed on 21/02/2021), pp. 267–277. issn: 00261270. doi: 10.1055/s-0038-1634340. url: http://www.thieme-connect.de/DOI/DOI?10.1055/s-0038-1634340.

Automate Your Laboratory with the Global Leader for LIMS and ELN (n.d.). (Accessed on 25/02/2021). url: https://www.labware.com/.

Bemmel, J. H. Van (July 1984). "The structure of medical informatics". In: *Informatics for Health and Social Care* 9 (3-4). (Accessed on 21/02/2021), pp. 175–180. issn: 17538157. doi: 10.3109/14639238409015187.

Bourgeois, David T., Ph.D., and Bourgeois (Feb. 2014). *Information Systems for Business and Beyond*. (Accessed on 20/02/2021).

Carr, Nicholas (Jan. 2007). *IT doesn't matter*. (Accessed on 20/02/2021). url: http://www.roughtype.com/?p=644.

Centers for Disease Control and Prevention (2021a). *Different COVID-19 Vaccines*. (Accessed on 03/02/2021). url: https://www.cdc.gov/coronavirus/2019-ncov/vaccines/different-vaccines.html.

– (2021). *How to Protect Yourself Others*. (Accessed on 03/02/2021). url: https://www.cdc.gov/coronavirus/2019-ncov/prevent-getting-sick/prevention.html?CDC_AA_refVal=https%3A%2F%2Fwww.cdc.gov%2Fcoronavirus%2F2019-ncov%2Fprepare%2Fprevention.html.

– (2021b). *Interim Guidance for Antigen Testing for SARS-CoV-2*. (Accessed on 04/02/2021). url: https://www.cdc.gov/coronavirus/2019-ncov/lab/resources/antigen-tests-guidelines.html.

COVID Response (n.d.). (Accessed on 25/02/2021). url: https://livehealth.in/covidResponse/.

Daemen, Joan and Vincent Rijmen (Sept. 1999). "AES Proposal: Rijndael". In: (Accessed on 29/09/2021).

Davis, Fred D. (1989). "Perceived usefulness, perceived ease of use, and user acceptance of information technology". In: *MIS Quarterly: Management Information Systems* 13 (3). (Accessed on 21/09/2021), pp. 319–339. doi: 10.2307/249008.

Diário de Notícias (Jan. 2021). *Portugal está a realizar uma média de 52 mil testes diários à covid-19*. (Accessed on 16/02/2021). url: https://www.dn.pt/sociedade/portugal-esta-a-realizar-uma-media-de-52-mil-testes-diarios-a-covid-19-13295199.html.

Direção-Geral da Saúde (2021). *COVID-19*. (Accessed on 16/02/2021). url: https://covid19.min-saude.pt/.

Dowdy, David and Gypsyamber D'Souza (2020). *COVID-19 Testing: Understanding the "Percent Positive" - COVID-19 - Johns Hopkins Bloomberg School of Public Health*. (Accessed on 16/02/2021). url: https://www.jhsph.edu/covid-19/articles/covid-19-testing-understanding-the-percent-positive.html.

Dworkin, Morris et al. (2001-11-26 2001). "Advanced Encryption Standard (AES)". In: (Accessed on 29/09/2021). doi: https://doi.org/10.6028/NIST.FIPS.197.

Evans, Eric (Mar. 2015). *Domain--Driven Design Reference Definitions and Pattern Summaries*. (Accessed on 28/02/2021). url: http://creativecommons.org/licenses/by/4.0/.ii.

Fichman, Robert G., Rajiv Kohli, and Ranjani Krishnan (Sept. 2011). *The role of information systems in healthcare: Current research and future trends*. (Accessed on 23/02/2021). doi: 10.1287/isre.1110.0382.

Fowler, Martin (Dec. 2005). *CommandQuerySeparation*. (Accessed on 25/09/2021). url: https://martinfowler.com/bliki/CommandQuerySeparation.html.

– (July 2011). *CQRS*. (Accessed on 25/09/2021). url: https://martinfowler.com/bliki/CQRS.html.

Fowler, Martin et al. (Nov. 2002). *Patterns of Enterprise Application Architecture*. (Accessed on 26/09/2021). Addison-Wesley Professional. isbn: 9780321127426.

Friedman, Bruce (Nov. 2008). *LIS vs. LIMS: It's Time to Blend the Two Types of Lab Information Systems*. (Accessed on 27/02/2021). url: https://labsoftnews.typepad.com/lab_soft_news/2008/11/liss-vs-limss-its-time-to-consider-merging-the-two-types-of-systems.html.

Gamma, Erich et al. (Oct. 1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. (Accessed on 25/09/2021). USA: Addison-Wesley Longman Publishing Co., Inc. isbn: 0201633612.

Glinz, Martin (2007). "On Non-Functional Requirements". In: *15th IEEE International Requirements Engineering Conference*. (Accessed on 02/03/2021). doi: 10.1109/RE.2007.45.

Hambling, Brian and Pauline van Goethem (May 2013). *User Acceptance Testing - A step-by-step guide*. (Accessed on 10/10/2021). BCS Learning Development Limited. isbn: 9781306283915.

Hauser, John R. and Don Clausing (May 1988). *The House of Quality*. (Accessed on 18/02/2021). url: https://hbr.org/1988/05/the-house-of-quality.

Haux, Reinhold (2010). "Medical informatics: Past, present, future". In: *International Journal of Medical Informatics* 79.9. (Accessed on 21/02/2021), pp. 599–610. issn: 1386-5056. doi: https://doi.org/10.1016/j.ijmedinf.2010.06.003. url: https://www.sciencedirect.com/science/article/pii/S1386505610001140.

Heeks, Richard (Feb. 2006). "Health information systems: Failure, success and improvisation". In: *International Journal of Medical Informatics* 75 (2). (Accessed on 23/02/2021), pp. 125–137. issn: 13865056. doi: 10.1016/j.ijmedinf.2005.07.024. url: https://linkinghub.elsevier.com/retrieve/pii/S1386505605001255.

Heeks, Richard, David Mundy, and Angel Salazar (Jan. 1999). "Why Health Care Information Systems Succeed or Fail". In: *SSRN Electronic Journal*. (Accessed on 23/02/2021). issn: 1556-5068. doi: 10.2139/ssrn.3540062.

Khorikov, Vladimir (Jan. 2020). *Unit Testing Principles, Practices, and Patterns*. (Accessed on 09/10/2021). Manning Publications. isbn: 9781638350293.

Koen, P. et al. (2001). "Providing clarity and a common language to the "fuzzy front end"". In: *Research Technology Management* 44 (2). (Accessed on 08/02/2021), pp. 46–55. issn: 08956308. doi: 10.1080/08956308.2001.11671418. url: https://www.tandfonline.com/doi/abs/10.1080/08956308.2001.11671418.

Lab Software for Medical Diagnostics | CrelioHealth LIMS (n.d.). (Accessed on 25/02/2021). url: https://livehealth.in/.

Ledley, Robert S. and Lee B. Lusted (July 1959). "Reasoning foundations of medical diagnosis". In: *Science* 130 (3366). (Accessed on 21/02/2021), pp. 9–21. issn: 00368075. doi: 10.1126/science.130.3366.9. url: https://pubmed.ncbi.nlm.nih.gov/13668531/.

Martin, Robert C. (Aug. 2012). *The Clean Architecture*. (Accessed on 05/03/2021). url: https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html.

Microsoft (Nov. 2009). *Microsoft® Application Architecture Guide, 2nd Edition (Patterns Practices)*. Second. (Accessed on 05/03/2021). url: https://www.amazon.com/Microsoft%C2%AE-Application-Architecture-Patterns-Practices/dp/073562710X.

– (June 2017). *Pipes and Filters pattern - Cloud Design Patterns*. (Accessed on 25/09/2021). url: https://docs.microsoft.com/en-us/azure/architecture/patterns/pipes-and-filters.

– (Jan. 2019). *Introduction to ASP.NET Identity*. (Accessed on 06/03/2021). url: https://docs.microsoft.com/en-us/aspnet/identity/overview/getting-started/introduction-to-aspnet-identity.

– (Jan. 2020a). *Common web application architectures*. (Accessed on 05/03/2021). url: https://docs.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/common-web-application-architectures.

– (Feb. 2020b). *Overview of ASP.NET Core MVC*. (Accessed on 06/03/2021). url: https://docs.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-5.0.

– (Feb. 2020c). *Overview of ASP.NET Core MVC*. (Accessed on 27/09/2021). url: https://docs.microsoft.com/pt-pt/aspnet/core/mvc/overview?WT.mc_id=dotnet-35129-website&view=aspnetcore-5.0.

– (July 2021). *Unit testing fundamentals*. (Accessed on 09/10/2021). url: https://docs.microsoft.com/en-us/visualstudio/test/unit-test-basics?view=vs-2019.

Nicola, Susana, Eduarda Pinto Ferreira, and J. J.Pinto Ferreira (July 2012). "A novel framework for modeling value for the customer, an essay on negotiation". In: *International Journal of Information Technology and Decision Making* 11 (3). (Accessed on 08/02/2021), pp. 661–703. issn: 02196220. doi: 10.1142/S0219622012500162.

Oliver, Sara E. et al. (Dec. 2020). "The Advisory Committee on Immunization Practices' Interim Recommendation for Use of Pfizer-BioNTech COVID-19 Vaccine — United States, December 2020". In: *MMWR. Morbidity and Mortality Weekly Report* 69 (50). (Accessed on 03/02/2021), pp. 1922–1924. issn: 0149-2195. doi: 10.15585/mmwr.mm6950e2. url: http://www.cdc.gov/mmwr/volumes/69/wr/mm6950e2.htm?s_cid=mm6950e2_w.

– (Jan. 2021). "The Advisory Committee on Immunization Practices' Interim Recommendation for Use of Moderna COVID-19 Vaccine — United States, December 2020". In: *MMWR. Morbidity and Mortality Weekly Report* 69 (5152). (Accessed on 03/02/2021), pp. 1653–1656. issn: 0149-2195. doi: 10.15585/mmwr.mm695152e1. url: http://www.cdc.gov/mmwr/volumes/69/wr/mm695152e1.htm?s_cid=mm695152e1_w.

Osterwalder, A. et al. (2014). *Value Proposition Design: How to Create Products and Services Customers Want*. The Strategyzer Series. (Accessed on 16/02/2021). Wiley. isbn: 9781118968055. url: https://books.google.pt/books?id=LCmtBAAAQBAJ.

Prokosch, Hans Ulrich and T. Ganslandt (Jan. 2009). "Perspectives for medical informatics". In: *Methods of Information in Medicine* 48 (1). (Accessed on 21/02/2021), pp. 38–44. issn: 00261270. doi: 10.3414/ME9132. url: http://www.thieme-connect.de/DOI/DOI?10.3414/ME9132.

Reeves, J. Jeffery et al. (June 2020). "Rapid response to COVID-19: Health informatics support for outbreak management in an academic health system". In: *Journal of the*

American Medical Informatics Association 27 (6). (Accessed on 16/02/2021), pp. 853–859. issn: 1527974X. doi: 10.1093/jamia/ocaa037. url: https://pubmed.ncbi.nlm.nih.gov/32208481/.

Sami, Hamid R. (Jan. 2019). *Medical Informatics in Neurology*. (Accessed on 21/02/2021). url: https://emedicine.medscape.com/article/1136989-overview.

SampleManager LIMS Software (n.d.). (Accessed on 25/02/2021). url: https://www.thermofisher.com/pt/en/home/digital-solutions/lab-informatics/samplemanager-lims.html#.

Sepulveda, Jorge L. and Donald S. Young (Aug. 2013). *The ideal laboratory information system*. (Accessed on 27/02/2021). doi: 10.5858/arpa.2012-0362-RA.

Wang, Yanrong et al. (May 2020). "Clinical Outcomes in 55 Patients With Severe Acute Respiratory Syndrome Coronavirus 2 Who Were Asymptomatic at Hospital Admission in Shenzhen, China". In: *The Journal of Infectious Diseases* 221 (11). (Accessed on 03/02/2021), pp. 1770–1774. issn: 0022-1899. doi: 10.1093/infdis/jiaa119. url: https://academic.oup.com/jid/article/221/11/1770/5807958.

Weemaes, Matthias et al. (Aug. 2020a). "Laboratory information system requirements to manage the COVID-19 pandemic: A report from the Belgian national reference testing center". In: *Journal of the American Medical Informatics Association* 27 (8). (Accessed on 25/02/2021), pp. 1293–1299. issn: 1527-974X. doi: 10.1093/jamia/ocaa081. url: https://academic.oup.com/jamia/article/27/8/1293/5827002.

– (Aug. 2020b). "Laboratory information system requirements to manage the COVID-19 pandemic: A report from the Belgian national reference testing center". In: *Journal of the American Medical Informatics Association* 27 (8). (Accessed on 16/02/2021), pp. 1293–1299. issn: 1527-974X. doi: 10.1093/jamia/ocaa081. url: https://academic.oup.com/jamia/article/27/8/1293/5827002.

Westbrook, Johanna I., Andrew Georgiou, and Mary Lam (2009). "Does computerised provider order entry reduce test turnaround times? A before-and-after study at four hospitals". In: vol. 150. (Accessed on 27/02/2021). IOS Press, pp. 527–531. isbn: 9781607500445. doi: 10.3233/978-1-60750-044-5-527. url: https://researchers.mq.edu.au/en/publications/does-computerised-provider-order-entry-reduce-test-turnaround-tim.

Willems, Jos L., Cassiano Abreu-Lima, et al. (Dec. 1991). "The Diagnostic Performance of Computer Programs for the Interpretation of Electrocardiograms". In: *New England Journal of Medicine* 325 (25). (Accessed on 21/02/2021), pp. 1767–1773. issn: 0028-4793. doi: 10.1056/NEJM199112193252503. url: http://www.nejm.org/doi/abs/10.1056/NEJM199112193252503.

Willems, Jos L., Pierre Arnaud, et al. (1985). "Establishment of a reference library for evaluating computer ECG measurement programs". In: *Computers and Biomedical Research* 18.5. (Accessed on 21/02/2021), pp. 439–457. issn: 0010-4809. doi: https://doi.org/10.1016/0010-4809(85)90021-7. url: https://www.sciencedirect.com/science/article/pii/0010480985900217.

Woodall, Tony (Jan. 2003). "Conceptualising 'Value for the Customer': An Attributional, Structural and Dispositional Analysis". In: *Academy of Marketing Science Review* 12. (Accessed on 09/02/2021).

World Health Organization (Oct. 2020). *Coronavirus disease (COVID-19)*. (Accessed on 03/02/2021). url: https://www.who.int/emergencies/diseases/novel-coronavirus-2019/question-and-answers-hub/q-a-detail/coronavirus-disease-covid-19.

– (Mar. 2020). *Laboratory testing for coronavirus disease 2019 (COVID-19) in suspected human cases: interim guidance, 2 March 2020*. (Accessed on 03/02/2021). url: https:

/ / apps . who . int / iris / bitstream / handle / 10665 / 331329 / WHO - COVID - 19 - laboratory-2020.4-eng.pdf?sequence=1&isAllowed=y.

Wu, Zunyou and Jennifer M. McGoogan (Apr. 2020). *Characteristics of and Important Lessons from the Coronavirus Disease 2019 (COVID-19) Outbreak in China: Summary of a Report of 72314 Cases from the Chinese Center for Disease Control and Prevention*. (Accessed on 03/02/2021). doi: 10.1001/jama.2020.2648. url: https://jamanetwork.com/.

Young, Ralph R. (2004). *The Requirements Engineering Handbook*. (Accessed on 01/03/2021).

Zairi, Mohamed and Mohamed A. Youssef (1995). "Quality function deployment: A main pillar for successful total quality management and product development". In: *International Journal of Quality Reliability Management* 12 (6). (Accessed on 18/02/2021), pp. 9–23. issn: 0265671X. doi: 10.1108/02656719510089894.

Zeithaml, Valarie (July 1988). "Consumer Perceptions of Price, Quality and Value: A Means-End Model and Synthesis of Evidence". In: *Journal of Marketing* 52. (Accessed on 09/02/2021), pp. 2–22. doi: 10.1177/002224298805200302.

Zitek, Tony (2020). *The appropriate use of testing for Covid-19*. (Accessed on 03/02/2021). doi: 10.5811/westjem.2020.4.47370. url: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7234686/.

Zwass, Vladimir (Nov. 2020). *information system*. (Accessed on 20/02/2021). url: https://www.britannica.com/topic/information-system.