NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

HENRIQUE FERREIRA COSTA JOAQUIM

BSc in Electrical and Computer Engineering

SYSTEM ARCHITECTURE AND WEB DEVELOPMENT FOR HEALTHCARE BIG DATA DRIVEN APPLICATION

ELECTRICAL AND COMPUTER ENGINEERING

NOVA University Lisbon

11, 2021

NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

SYSTEM ARCHITECTURE AND WEB DEVELOPMENT FOR HEALTHCARE BIG DATA DRIVEN APPLICATION

HENRIQUE FERREIRA COSTA JOAQUIM

BSc in Electrical and Computer Engineering

Adviser: João Paulo Branquinho Pimentão
Auxiliar Professor, NOVA University Lisbon

Co-adviser: Pedro Alexandre Sousa
Associate Professor, NOVA University Lisbon

Examination Committee:

Chair: Paulo Miguel de Araújo Borges Montezuma de Carvalho
Associate Professor, NOVA University Lisbon

Rapporteur: João Almeida das Rosas
Auxiliar Professor, NOVA University Lisbon

Adviser: João Paulo Branquinho Pimentão
Auxiliar Professor, NOVA University Lisbon

ELECTRICAL AND COMPUTER ENGINEERING

NOVA University Lisbon
11, 2021

System Architecture and Web Development for Healthcare Big Data Driven Application

Para os meus pequenos sobrinhos.

Acknowledgements

First, I would like to thank my supervisors, João Paulo Pimentão and Pedro Sousa; in addition to my teachers, they became my friends and people I take as an example. Thank you for believing that this job was possible and for not letting me give up, also for having included me in the Clarify project and allowing me to be part of a project that positively impacts people's lives.

To FCT NOVA for having been my home for the last 5 years and that, in fact, it will always be.

To NEEC, where I learned a lot and made great friends. It was an immense pleasure to be part of what is the best student group of the Faculty.

To In-Nova, the junior enterprise of FCT NOVA, and all its members for the fantastic experience, which made me grow so much. Especially Joana Caetano, Filipa Rebelo, and Telma Esteves, with whom I had the privilege of spending so many hours and learning so much.

To all my HOLOS colleagues with whom I have learned so much, particularly Sofia Rodrigues, with whom I started this journey, thanks for always being ready to help. And to Alexandre Sousa, who was a tireless partner in developing this project.

To my training partners and teachers at Rubik in Almada and at Vita Team in Évora, jiu-jitsu played a central role in my life and taught me a lot. For that reason, my thanks to everyone involved in this journey.

To my dear friends and classmates, Bruno Vieira, Francisco Róis, Serafim Ciobanu and Nikita Dyskin. Without you, it would not have been possible to go this far. Thank you for always being with me and for being the fantastic people that you are.

To my childhood friends, the "Irmandade," who have always played a central role in my life, may we have many more years of adventures.

To my uncles, tia Isabel and tio Leonardo, for being wonderful and kind and treating me so well. To my cousins, Ricardo and Cláudia, who are like brothers and who fill me with pride for the fantastic people they have become.

To my mom and dad for always believing in me so much, always giving me all the support and some more without hesitation, there are not enough words to thank you, and

I hope I have become a person you are proud of.

To my sister and Louis, who put up with me for two years and continue to do so every time needed. Thank you, mana, for always being with me and always ready to defend me in any circumstance. I hope someday to be at your level.

To my little nephews, Leonor and Miguel, for being the most incredible light of my life. I hope someday I can give you the world.

Finally, to Alexandra Pimenta, the person who knows me best and with whom I shared the best and worst moments and who was the closest person to me in the last 6 years. There are no words to describe what we went through together, and it certainly left deep marks on the people we became. Thank you for everything. My success is largely due to you.

"All things are difficult before they are easy."
Thomas Fuller

Abstract

With the current increase in the volume, variety, and complexity of data, Big Data is increasingly becoming a needed paradigm in any sector of activity. Given these facts, the need for computer systems capable of responding to these same data, especially in processing, storage, and presentation is increasing. All of these points are fundamental so that it becomes to work with the data in a way that it is possible to extract value and knowledge from it; whether to intensify productivity on an assembly line, increase a business' revenue, or improve the quality of life of a given population.

The question then arises of how we can develop such computer systems in the context of Big Data applied to the healthcare sector. To respond to the challenges imposed by this scenario, it is necessary to integrate multiple data sources, process and present them to the end-user in an understandable and timely manner so that their use is viable.

As a solution proposal, a system architecture based on microservices is presented, in which the presentation of data uses the latest Web development tools. Such an architecture uses a Cloud infrastructure to take advantage of the inherent advantages, such as scalability, security, and flexibility.

From the analysis of data from different sources, with various clinical practices which add volume on which to infer, it is expected that advanced data processing techniques will support the development of new treatment methodologies, support current methods, or even create fertile ground for the creation of practices that could improve the quality of oncological patients.

Keywords: Big data, Microservices architecture, Data processing, Data analysis, Cloud Computing, Web development

Resumo

Com o aumento no volume, variedade e complexidade dos dados, cada vez mais se caminha para um paradigma de Big data em todo e qualquer sector de ativade. Perante tais factos, surge cada vez mais a necessidade de existirem sistemas informáticos capazes de dar resposta às necessidades impostas por estes mesmos dados, sobretudo em aspetos como o seu processamento, armazenamento e apresentação. Todos estes pontos são fundamentais para que seja possível trabalhar os dados de forma a que se possa extrair valor e conhecimento destes, seja com vista à intensificação da produtividade numa linha de montagem, aumento das receitas de um negócio ou melhorar a qualidade de vida de uma determinada população.

Surge então a questão de como podemos desenvolver tais sistemas informáticos num contexto de Big data aplicado ao sector da saúde. Tendo em conta a forma de responder aos desafios impostos, é necessária a integração de múltiplas fontes de dados assim como a capacidade de as tratar e apresentar ao utilizador final de forma compreensível e em tempo útil, para que a sua utilização seja viável.

Como proposta de solução, é proposta uma arquitetura de sistema baseada em microsserviços em que a apresentação dos dados recorre às mais recentes ferramentas de desenvolvimento Web. Tal arquitetura serve-se da infraestrutura Cloud por forma a tirar partido das vantagens inerentes à mesma, tais como escalabilidade, segurança e flexibilidade.

Com a análise e integração de diferentes fontes de dados e recorrendo a técnicas avançadas de processamento de dados, é esperado que sejam oferecidas novas perspetivas que possam apoiar o desenvolvimento de novos métodos de tratamento, adoptar aqueles que já existam e criar solo fértil para a criação de novas práticas, em que o objetivo passa por melhorar a qualidade de vida de pacientes oncológicos.

Palavras-chave: Big data, Arquitetura de microsserviços, Análise e integração de múltiplas fontes de dados, Cloud Computing, Desenvolvimento Web

Contents

List of Figures xii

List of Tables xiv

Glossary xvi

Acronyms xvii

1 Introduction 1
- 1.1 Background . . . 1
- 1.2 Research Problems . . . 2
- 1.3 Main Contributions . . . 3
- 1.4 Dissertation Outline . . . 4

2 Related Work 5
- 2.1 Big data in healthcare . . . 5
  - 2.1.1 Characteristics of Big data . . . 6
- 2.2 Big data frameworks . . . 6
  - 2.2.1 Hadoop . . . 6
  - 2.2.2 Apache Spark . . . 7
  - 2.2.3 Cloud Computing . . . 8
  - 2.2.4 Serverless Clouds . . . 10
- 2.3 Data Storage . . . 13
- 2.4 Data Management . . . 15
  - 2.4.1 Extract Transform and Load . . . 15
- 2.5 Data Visualisation and Reporting . . . 18

3 Architecture and Methodology 19
- 3.1 Technology Stack . . . 19
- 3.2 System Requirements . . . 20
- 3.3 Architecture Overview . . . 21

3.4 End-user Interaction . . . 23
3.5 Data Sources Integration Process . . . 25
3.6 Models Integration Process . . . 27

4 Implementation **29**
4.1 Infrastructure . . . 29
4.1.1 Google Cloud Platform Overview . . . 30
4.1.2 Project Creation . . . 32
4.1.3 App Engine . . . 32
4.1.4 Compute Engine - Virtual Machine . . . 33
4.1.5 Cloud Storage . . . 38
4.1.6 Cloud SQL . . . 38
4.1.7 Google Cloud Functions . . . 39
4.1.8 Firebase . . . 43
4.2 Data Sources . . . 46
4.2.1 Population Data . . . 47
4.2.2 Kronohealth files . . . 50
4.2.3 Medical questionnaire . . . 52
4.2.4 Container optimised VM . . . 53
4.3 Services - Google Cloud Functions . . . 54
4.3.1 Google Cloud Functions . . . 54
4.3.2 Google Cloud Functions optimisation . . . 55
4.4 User Interface . . . 57

5 Results **61**
5.1 Google Cloud Functions Optimisation Results . . . 61
5.2 Web Application Results Overview . . . 64

6 Conclusion and Future Work **68**
6.1 Future Work . . . 69

Bibliography **71**

Annexes

I Pentaho Extract Transform and Load (ETL) Tool Transformations **77**

II Google Cloud Functions Optimisation Results **81**

List of Figures

1.1 Big data analysis process, adapted from [61]. 3

3.1 Big data-driven Architecture Overview. 22
3.2 User interaction flowchart. 24
3.3 Data source integration process flowchart. 26
3.4 Models integration process flowchart. 28

4.1 Project information on Google Cloud Platform (GCP). 30
4.2 Google Cloud Console example. 31
4.3 Cloud Shell (gcloud command-line) interface example. 31
4.4 Initial page of GCP after setting up an account. 32
4.5 Create instance menu. 34
4.6 Configured Virtual Machine (VM) ready for usage. 34
4.7 Configured VM Secure Shell (SSH) interface with typical Linux filesystem. 35
4.8 Setting up a new firewall rule. 36
4.9 VM User Interface (UI) on Virtual Network Computing (VNC) Viewer. . . 36
4.10 VM UI on VNC Viewer. 37
4.11 Available Structured Query Language (SQL) versions to create a Cloud SQL instance. 38
4.12 Created MySQL instance. 39
4.13 Example of a Cloud Function Configuration. 40
4.14 GCF development environment with a basic example. 41
4.15 Example of Google Cloud Functions (GCF) invocation through Postman. . 42
4.16 Firebase console. 43
4.17 Authentication methods on Firebase. 44
4.18 User registration on Firebase. 44
4.19 Example of a menu in Cloud Firestore. 44
4.20 Rule definition example on Firebase. 45
4.21 Database Schema Model on MySQL Workbench. 48
4.22 ETL Job in Pentaho. 50

4.23 File structure of Kronohealth files. 51
4.24 Container Registry menu with successfully imported Docker image. 53
4.25 Verifying container optimised Operating System (OS) VM functionality. . . 54
4.26 UI organization. 58

5.1 Results on cold start and warm response before and after the optimisation for the survival_curve Cloud Function. 62
5.2 Results on cold start and warm response before and after the optimisation for the dataset_info Cloud Function. 63
5.3 Results on cold start and warm response before and after the optimisation for the patient_info Cloud Function. 63
5.4 Login page. 64
5.5 Homepage. 65
5.6 Individual Analysis 66
5.7 Individual Analysis 67

I.1 Patients table transformation. 77
I.2 Radiotherapy table transformation. 78
I.3 Treatments table transformation. 78
I.4 Treatments table transformation. 79
I.5 Surgery table transformation. 79
I.6 Progression table transformation. 80

II.1 dataset_info cold start optimisation results. 81
II.2 dataset_info warm response optimisation results. 81
II.3 patient_info cold start optimisation results. 82
II.4 patient_info warm response optimisation results. 82
II.5 survival_curve cold start optimisation results. 82
II.6 survival_curve warm response optimisation results. 82

List of Tables

2.1 Main features of leading Cloud providers, based on [47], [48], [2], [5] and [13]. 12
2.2 Cloud Storage Platforms, based on [61]. 14
2.3 Open-source ETL tools, based on [65], [67], [43] and [57]. 17
2.4 Data Visualisation JavaScript frameworks. 18

List of Listings

4.1 Deploy application on localhost. 32
4.2 Deploy application to App Engine. 33
4.3 Setup VNC. 35
4.4 Setup VNC. 37
4.5 Creating a Cloud Storage bucket. 38
4.6 Deploy example Cloud Function. 42
4.7 Example of a menu organisation as a JavaScript Object Notation (JSON) object. 45
4.8 Example of a rule definition as a JSON object. 46
4.9 File upload to bucket. 52
4.10 Example of database lazy loading. 57

Glossary

GB-second Total amount of memory allocated to a process per second that that task runs. 12

multitenancy architecture A software architecture in which a single instance of software operates on a server and serves numerous tenants. Such systems are referred to as "shared." A tenant is a set of users who have access to the same program instance but have different privileges. 8

Acronyms

AI Artificial Inteligence 10
API Application Programming Interface 10, 15, 25, 30, 31, 40, 54, 55, 59
AWS Amazon Web Services 9, 10
AWS SNS Amazon Simple Notification Service 10

CPU Central Processing Unit 7, 10
CSV Comma-separated Values 51, 60

EHR Electronic Health Record 5
ETL Extract Transform and Load xi, xii, xiv, 15, 16, 17, 20, 21, 23, 27, 33, 35, 46, 49, 50, 69, 77

FaaS Function-as-a-Service 54

GCF Google Cloud Functions xii, 21, 27, 39, 41, 42, 54, 55, 56, 61, 62, 69
GCP Google Cloud Platform xii, 9, 10, 19, 20, 29, 30, 32, 33, 34, 38, 41, 43, 53, 54, 69

HDFS Hadoop Distributed File System 7
HTTP Hypertext Transfer Protocol 10, 40

ICT Information and Communications Technology 8
IoT Internet of Things 9
IP Internet Protocol 39, 53

JS JavaScript 20
JSON JavaScript Object Notation xv, 44, 45, 51, 53, 54, 55

OS Operating System xiii, 53, 54

RDBMS Relational Database Management System 19, 38

SDK Software Development Kit 31
SMEs Small and medium-sized enterprises 10
SQL Structured Query Language xii, 7, 19, 29, 38
SSH Secure Shell xii, 34, 35

UI User Interface xii, 21, 23, 29, 32, 33, 35, 36, 37, 43, 44, 54, 55, 57
UX User Experience 21, 23, 27, 49, 57, 61

VM Virtual Machine xii, xiii, 30, 33, 34, 35, 36, 37, 53, 54, 56
VNC Virtual Network Computing xii, xv, 35, 36, 37
VPC Virtual Private Cloud 34, 39, 53

YARN Yet Another Resource Negotiator 7

1

Introduction

This chapter aims to present the proposed work. It begins with the background, which represents a motivation for the development of the project. Then the research approach is described and it is presented the research problem as well as the hypothetical solution. Next in order are discussed the potential contributions to the scientific community. Finally, the structure of the document is presented.

1.1 Background

As of 2012, the digital world of data was expanded to 2.72 zettabytes[1] by 2024 the amount of data is expected to reach 149 zettabytes [40] and 0.463 zettabytes will be produced everyday by 2025 [10]. Big enterprises are collecting trillions of bytes of data regarding their customers, suppliers and operations per hour, as it offers promise in several business sectors to answer many issues [45].

With this kind of increase in the volume and variety of data in different sectors, it is necessary to process and manage that data conveniently and adequately for their intended purpose. Therefore, this implies the use of new tools, technologies and approaches for curating, storage and processing [11], in healthcare, this need is no different [18].

In this same context and aimed at healthcare, there is even another kind of challenge. According to [45] and [11], the main concerns are:

1. Data structure (fragmented data, incompatible formats, raw and unstructured datasets);

2. Security (privacy, confidentiality, data duplication, integrity);

3. Data standardisation (limited interoperability, data acquisition and cleansing and language barriers);

4. Storage and transfers (expensive to store, securely extract, transmit and process).

[1] 1 zettabyte = $1024^7 bytes$

Although many challenges, there are also crucial factors that drive Big data usage and research in this field, according to [50]:

1. Clinical decision support systems;
2. Individual analytics applied for patient profile,
3. Personalised medicine,
4. Performance-based pricing for personnel,
5. Analyse disease patterns,
6. Improve public health.

Concerning the healthcare sector an enormous volume and variety of data is obtained from several different sources, such as Electronic Health Records, monitoring devices, health trackers and intelligent devices, medical procedures - treatments, imaging or surgery. As so, volume and variety are two of the main concepts that contribute to coining the term Big data [17]. This problem becomes as complex as the amount of units in consideration increase. If the objective is to collect and process data not only from a healthcare unit but from a group of these at local, regional, national, European or global levels. From the analysis of data from different sources, with various clinical practices which add volume on which to infer, it is expected that advanced data processing techniques will support the development of new treatment methodologies, support current methodologies, or even create fertile ground for the creation of practices that could improve the quality of human life.

1.2 Research Problems

Even though there are well-known good reasons to dive deeper into Big data and its applications, finding the right approaches to achieve it is still a challenge, specifically to give a response to the issues mentioned above, and, for that reason, the most advanced health IT systems still rely on data warehouse structures.

Without the proper infrastructure, as well as the appropriate analytic tools and visualisation approaches, the insight provided by Big data is likely to be limited [60].

As proposed by [61], Big data healthcare analytics has five processes:

1. Data Acquisition;
2. Data Storage;
3. Data Management;
4. Data Analytics;

5. Data Visualisation and Reporting.

Figure 1.1 shows the flow of data throughout a data-centred application.

Figure 1.1: Big data analysis process, adapted from [61].

This work aims to build a valuable and accessible framework for healthcare professionals to work with and make decisions, considering the clinical context. It is necessary to address the processes mentioned earlier as different problems, considering the system's interoperability as a whole.

As healthcare's Big data exceeds the ability of standard data processing techniques, we must take to other approaches such as distributed computation [18] or cloud computing.

Considering the challenges that arise from the current status quo described above, it should be clear that new solutions are needed. As so, the following research problem is posed:

- How can Cloud computing support Big data Web applications to assist healthcare professionals in a clinical scenario?

As a hypothetical solution to this problem, this work proposes the use of Google Cloud Platform, a suite of Google's public cloud computing resources and services, serverless architecture, to create a Web application capable of handling Big data in healthcare. In this work, a serverless architecture will be developed to integrate multiple data sources, addressing the storage and management of that data. Additionally, a Web application, which is the final product, will be implemented.

1.3 Main Contributions

The proposed problem consists of developing software capable of responding to the challenges imposed by Big data and implementing a Web application to support it. Therefore, we can divide the problem into the following subsections:

1. Overall Systems Architecture;
2. Data Acquisition and Data Storage;
3. Data Management or Analytics;
4. Data Visualisation and Reporting.

Although it is possible to think of these subsections of distinct and separate problems requiring different implementation, the solution should consider the system interoperability across the various components.

The architecture proposed in this project is aimed to be applied to the healthcare sector. However, any other could benefit from it with the proper adjustments, as Big data can be found in every sector of the global economy [50]. Regarding the Web application developed, this only applies to healthcare purposes. However, it serves the implementation presented here as a guideline for implementing applications for other sectors.

Additionally, the studies performed herein demonstrate the usefulness and performance of the Google Cloud Platform in actual use cases. The optimisation of Google Cloud Functions is further discussed and opportunities to improve, allowing future approaches to consider this feature to obtain better results.

The Google Cloud Platform and React-Redux Javascript framework proved to be very useful and capable, thus encouraging further research to develop Big data platforms.

This thesis is also part of the European research project Clarify[2] which proposes integrating and analysing vast quantities of heterogeneous multivariate data to aid in the early detection of risk factors that may worsen a patient's condition after oncological therapy has ended. This will successfully help to stratify cancer survivors by risk so that their follow-up can be tailored to their specific needs.

1.4 Dissertation Outline

Initially, it introduces the background, which serves as motivation for the research developed in this document. Then the research problem is presented as well as the hypothetical solution and potential contributions.

The Related Work section presents the state of the art regarding Big data, Cloud computing, and related technologies such as Data Storage, Data Management, and Data Visualisation and Reporting. After analysis of the previous section, the technology stack to develop the work in hand and proposed architecture to address the research are discussed and presented.

A methodology is proposed for implementing the data sources, the developed services, and the presupposed user interaction with the system.

The implementation of the proposed work explains the infrastructure, data sources, services, and user interface. Lastly, the implementation results and the conclusion and future work are presented, where improvements to the developed application are discussed.

[2] https://www.clarify2020.eu

2

Related Work

2.1 Big data in healthcare

The practice of storing healthcare data in digital records is relatively new. The standard approach to keep patient information was either handwritten notes or typed reports [19], being the first occurrence of this practice found circa 1600 BC in a papyrus text from Egypt [20].

With globalisation, increased access to information systems and digitalisation, the storage of all clinical information in the healthcare systems became a standard practice [15].

Eventually, in 2003, the Institute of Medicine chose the term Electronic Health Record (EHR). EHR is defined by [59] as any information relating to an individual's physical or mental health condition that resides in an electronic system for the primary purpose of providing healthcare and health-related services.

EHR brought many advantages, according to [15]:

1. Ease of access to medical history;
2. The overcome of logistical errors;
3. Improvement of medical practices with automatic reminders and prompts;
4. Greater continuity of care and timely interventions;
5. Faster data retrieval and facilitate reporting of key healthcare quality indicators;
6. Relevant data regarding the quality of care;
7. Reduce delays and confusion in the billing and claims management area.

Besides the advantages described above, the EHR combined with the internet have been helping provide access to critical information for patient life and care [15].

With such benefits, it is easy to understand why a Big data scenario in healthcare has been reached nowadays, with lots of information being created every second regarding operations, staff and patient information from appointments to logistics or medical procedures, generating a vast universe of complex and difficult to analyse data.

2.1.1 Characteristics of Big data

Even though the relevance of big data has been well recognised, there is still no consensual definition of Big data, despite several authors having similar ideas. A report submitted by U. S. Congress in August 2012 explained big data as an enormous number of fast-moving, complex, and changing data that necessitate modern procedures and technology to gather, store, distribute, manage, and analyse the data [61]. Hence, the famous Big data "V's", which are not consensual, also being the most well known, defined as soon as 2001, the "3V's" [12]. With that being said, some of the essential Big data characteristics are:

1. Volume - generation and collection of immense amounts of data leads to data scale increasingly bigger.

2. Velocity - data is generated with such a pace that it requires distinct (distributed) processing techniques.

3. Variety - indicates the various types of data, including structured, semi-structured and unstructured data and the need to relate them.

4. Veracity - data can only add value if accurate and precise; thus, it can be proven.

5. Value - extracting hidden patterns from the data can lead to knowledge that can add significant value.

2.2 Big data frameworks

Some of the most popular frameworks for Big data processing include Hadoop, and Apache Spark [42]. In this section, its main features and corresponding architectures are presented.

2.2.1 Hadoop

Hadoop is a highly scalable open-source framework that allows the collection, processing, and storage of huge amounts of unstructured complex data distributed across clusters of computers so that they can work in parallel (hardware layer) [7] [37]. Hadoop is composed of four main layers, according to [11], [7] and [37]:

1. Core: a collection of standard utilities and libraries that support the other modules;

2. Distributed file system (Hadoop Distributed File System (HDFS)): provide high-performance access to complex data through a master-slave architecture where the master machine controls multiple slave devices providing a parallel orchestration as it divides the data through multiple nodes in a cluster;

3. Yet Another Resource Negotiator (YARN): a framework for job scheduling and resource management, such as Central Processing Unit (CPU) and memory;

4. Map-reduce: layer responsible for parallel processing of large datasets.

2.2.2 Apache Spark

Spark is an open-source unified data analytics cluster computing framework that is fast and powerful for large-scale data processing. Standing out, according to [1]:

1. Ease of use: the possibility to write applications in several programming languages;

2. Generality: the ability to combine a stack of analytics libraries;

3. Cross-platform: able to run in several platforms and access data from hundreds of data sources.

According to [7], typically, a Spark-based application have five layers:

1. Data storage systems: such as HDFS and HBASE (open-source non-relational distributed database [1]);

2. Resource management: such as YARN and Mesos (manage computer clusters abstracting computer resources away from machines [2]);

3. Spark Core Engine: acts as the application's kernel as it orchestrates the data processing and further operations;

4. Stream Processing: supports many applications and libraries such as Structured Query Language (SQL), MLib, R, GraphX and others;

5. Application program interface: usually written in a general-purpose programming language compatible with Spark: Java, Python or Scala.

The leading Public Cloud Providers have already adopted this design in their products and infrastructure.

[1] https://hbase.apache.org
[2] https://mesos.apache.org

2.2.3 Cloud Computing

To adequately process Big data, there is a need for high computing power clusters accessed via grid computing infrastructures capable of working in a distributed way; Cloud computing is such a system [15].

The objective of cloud computing is to use large amounts of computational resources, such as processing capacity and storage, under concentrated management so that it can provide applications with fine-grained computing capabilities [12].

Cloud computing has distributed storage technology that can efficiently manage vast amounts of data, and parallel computing power can improve the efficiency of data collecting and analysis. These platforms can operate as a data receiver from numerous sources, as an infrastructure to analyse and interpret the data, and as a web-based visualisation tool to provide the end-user with essential insights.

The use of Cloud Computing is a combination of a series of disciplines such as Infrastructure-as-a-Service, Platform-as-a-Service, and Software-as-a-Service. In the perspective of the user, it can be thought of as a Pay-per-Use-On-Demand convenience capable of addressing a series of complex tasks such as networking, computing power, storage, application, data analysis and so on [56].

According to [41] and [56] there are several implementation models which may be considered according to the user needs:

1. Public cloud - available to the public in general, it is controlled and managed by the organisation that contracts it from the provider; may be limited with security control over data and network [16].

2. Private cloud - only allows the company authorised staff to access the secure internal network. Services are not exposed to the public, meaning this solution can provide complete control. However, it may present more significant costs related to the data centre, cloud service management and hardware. May present fewer security-related challenges than Public cloud since the last has a multitenancy architecture, which increases risk [44].

3. Hybrid cloud - this model provides a combination of public and private cloud models together [6]; offers standard services such as sharing resources, storage space, and virtualisation.

2.2.3.1 Cloud Providers

Cloud computing models have brought significant advantages in the most varied branches of Information and Communications Technology (ICT) and, therefore, a paradigm shift, thus providing flexibility and benefits at the business level for companies that intend to use this model.

For this reason, the most prominent players in the market wanted to find the means to enter this field quickly.

Various cloud providers are competing in the market for their shares, such as Amazon Web Service, Microsoft Azure, IBM SmartCloud, Google Cloud Platform, and many more [44].

In this section, the three biggest cloud providers in the market will be presented and discussed:

- Amazon Web Services (AWS) [3]
- Microsoft Azure [4]
- Google Cloud Platform (GCP) [5]

The following characteristics are based on [4], [66] and [3]:

1. Amazon Web Services holds the largest market share. The vast toolset and depth of its services are pointed as the key strength for AWS. It also presents excellent developer functionality. This wide range of offers comes with the tradeoff of standard pricing challenges in understanding AWS metrics and estimating eventual costs.

   The main focus of AWS is the Public cloud, and it offers services such as:

   a) Compute;
   b) Databases and storage;
   c) App integration;
   d) Developer, management and engagement tools;
   e) Machine learning and predictive analytics;
   f) Business productivity tools.

2. Microsoft Azure is a preferred choice of enterprises with a previous relationship with Microsoft, presenting a combination of Azure, Teams, and Office 365, which may answer to a wide range of organisational challenges. Azure has built-in ready to run server apps that support programming languages such as .NET, Java, PHP, Node.js and Python.

   The main focus of Azure are Public and Hybrid cloud, and it offers services such as:

   a) DevOps;
   b) Blockchain technology;
   c) Internet of Things (IoT) integration;
   d) Big data and predictive analytics;

[3] https://docs.aws.amazon.com/
[4] https://docs.microsoft.com/en-us/azure
[5] https://cloud.google.com/why-google-cloud

e) Scalable data warehousing;

f) Game and app development.

3. Google Cloud Platform is a relatively new solution with proficiency in open-source technologies that values innovation. GCP draws corporate customers with their profound technical expertise and tools in artificial intelligence, machine and deep learning and data analytics. GCP has a wide range of serverless solutions and some of the most competitive pricing in the industry, which may be especially attractive for Small and medium-sized enterprises (SMEs).

   The main focus of GCP are Public and Hybrid cloud, and it offers services such as:

   a) Business analytics and Artificial Inteligence (AI);

   b) Application development;

   c) Data management and storage;

   d) Productivity and workload management tools.

2.2.4 Serverless Clouds

Serverless applications address Web and other distributed applications. It is a cloud computing execution model in which the cloud provider allocates machine resources on-demand, taking care of the servers on behalf of their customers. Therefore it allows programmers to focus on development and use a set of existing cloud services directly.

Examples of those services are Firebase (GCP) or DynamoDB (AWS), messaging systems such as GCP's Cloud Pub/Sub [6], notification services such as Amazon Simple Notification Service (AWS SNS) [7] and many others [48].

It is also convenient to add custom features recurring to "cloud functions", a subtype of Microservices' architecture, even more specialised, namely Function-as-a-Service (FaaS). It allows teams to work in complete abstraction, contributing to the end application. Examples of such functions include AWS Lambda [8], GCP's Google Cloud Functions [9] or Microsoft Azure Functions [10].

Those are based on functional programming and can be deployed on the providers' infrastructure. Functions can be triggered by: An event (database modification, file upload, scheduled action, between others); a direct Hypertext Transfer Protocol (HTTP) request or cloud Application Programming Interface (API) calls.

The cloud infrastructure is responsible for automatically provisioning resources, such as CPU, memory, network and storage, and automatic scaling of function executions according to the number of requests.

[6] https://cloud.google.com/pubsub/docs/overview
[7] https://aws.amazon.com/sns/
[8] https://aws.amazon.com/lambda/
[9] https://cloud.google.com/functions/
[10] https://docs.microsoft.com/en-us/azure/azure-functions/

In Table 2.1, the key features of the leading cloud function providers are presented.

	Google Cloud Functions	AWS Lambda	Azure Functions
Language	Node.js, Python, Go, Java, .NET, Ruby, and PHP	Java, Go, PowerShell, Node. js, C#, Python, and Ruby	C#, JavaScript, F#, Java, PowerShell, Python, TypeScript
Pricing	First 2 million are free and $0.4 beyond 2 million	$0.20 per 1 million requests and $0.00001667 for every GB-second used	$0.20 per 1 million requests and $0.000016 for every GB-second used
Triggers	HTTP, Cloud Storage, Cloud Pub/Sub, Cloud Firestore, Firebase, Cloud Logging	Lambda API, AWS SDK, AWS CLI and AWS toolkits	Queue, Timer, Event Grid, HTTP
Deployment	Zip and CVS	Container images and .zip file archives	Zip and CVS
Versioning	Possible	Possible	Possible
Dependency management	Yes, according to language	Yes, according to language	Yes, according to language
Execution time	540s	300s (maximum execution time)	No limit
Disk space	Consumes memory resources (configurable on deploy)	512MB	5TB
Number of functions	1000	1024	10
Parallel Execution	400	100 functions in parallel (can be increased on request)	No limit

Table 2.1: Main features of leading Cloud providers, based on [47], [48], [2], [5] and [13].

2.3 Data Storage

Storage is a critical part to take into consideration when building a Big data application. As the data size in the healthcare industry increases day by day, an efficient, large and scalable storage platform is needed.

With such large volumes of data, the cloud is the most promising solution since it provides elasticity and potential for scalable analytics [61].

According to [38], cloud computing has infrastructure capable of addressing the data storage essential to accomplish Big data analysis.

Table 2.2 describes cloud storage platforms critical points of interest.

Type	Vendor	Components/Platform	Application
Bid data Storage	Google	Cloud Services (Big Table/ Cloud Storage)	Cloud usage for clinical applications with private or hybrid cloud provide maximum level of security, privacy and availability.
	Amazon	Cloud Services (Amazon S3, Amazon Elastic File System)	
	Microsoft	Azure S3	
Relational Databases	Google	Cloud SQL	Highly available SQL database application with automatic provisioning, management and provided as a cloud service.
	Amazon	Amazon Relational Database Service (RDS)	
	Microsoft	Microsoft Azure SQL Database	
NoSQL	Google	Firestore	Healthcare applications commonly have unstructured complex data making NoSQL databases an important tool regarding scalability and flexibility.
	Amazon	Dynamo DB	
	Microsoft	Cosmos DB	

Table 2.2: Cloud Storage Platforms, based on [61].

2.4 Data Management

Data management includes organising, cleaning, retrieval, mining, and data governance [61]. Thus, to build a practical Big data healthcare application, it is necessary to consider strict and dedicated Extract Transform and Load (ETL) processes and the whole life cycle of data. Then, arises the need to relate this type of process with data governance strictly.

As defined by [69], data governance is everything one does to keep data secure, private, correct, accessible, and usable.

It entails the activities that people must take, the processes they must follow, and the technology that will help them along the data life cycle.

2.4.1 Extract Transform and Load

ETL is a process that is used to extract complex data from different sources, transform that data into a suitable and ready to use format and load it into the final target [49].

The goal is to prepare data for analysis by making it accessible and relevant, aggregate it to analyse and drive business decisions. It can also be used for a variety of other activities [63], such as data cleansing, data integration, data warehousing and data migration.

Companies can benefit immensely from improved access to their data, regardless of their size, complexity, or number of data sources. ETL processes allow users to see what is going on in their organization's operations.

Because the findings are only as accurate as the input given, this process is critical and the cornerstone of data analysis-based applications [62].

ETL components are:

1. **Extract** - extracting data and metadata from different sources and systems inside and outside the organization's scope into consolidated formats.

2. **Transform** - transform the data into a suitable and ready to work with format. May involve applying business rules, cleaning data, filtering, splitting and merging, joining together data from different sources (lookup / merge), transposing rows or columns, applying verification and validation rules or even send data to APIs for further processing.

3. **Loading** - output the data to use sources such as data marts, data lakes, data warehouses, databases, files (.csv, .txt, .xls, .xml), other reporting applications or directly to operational applications.

The key types of ETL tools are:

1. **Third-party** - are the most common ETL tools used by the majority of companies. These tools are often built to scale and have a strong pool of developers that can build on them. The majority of them have a high-level drag and drop interface.

2. **Library-based** - those allow more flexibility and customization since they are built on programming languages, providing developers great tools for one specific purpose but taking the risk of lack of generalization.

Also, those can be divided taking into account their commercial purpose, being either Commercially distributed or Open-source.

When choosing the adequate tool, according to business requirements, it is important to consider the following criteria: according to [62], [46] and [49]:

1. Supported connectivities (such as different database services providers);
2. Types of data transformation;
3. Delivery support;
4. Data modelling;
5. Data governance support;
6. Debugging facility;
7. Execution and runtime metrics;
8. Usability and integration;
9. Performance;
10. Customer support;
11. Documentation;
12. Ease of use;
13. Cost (hardware, software and other).

Since there are multiple offers in terms of ETL tools and this document is written for academic purposes, the open-source tools will be given priority.

As so, Table 2.3 intends to present some of the most relevant features of the two most suggested and accessible open-source applications based on [55], [62], [21] and [64].

Tool	Big Data Integration Pug-ins	Business Applications Integration	Support for analytics and Data Mining	RDBMS	NoSQL databases and object stores	Files
Pentaho Data Integration (PDI)	Cloudera, MapR (HPE Ezmeral Data Fabric), Amazon Web Services, Google Cloud and Microsoft Azure HDInsights	SAP, Salesforce and Google Analytics	R, Python, Scala and Weka	Oracle, IBM® DB2®, MySQL, Microsoft SQL Server	MongoDB, Cassandra, HBase, Hitachi Content Platform, AWS S3, Google Cloud Storage, Microsoft Azure ADLS Gen 2	XML, JSON, Microsoft Excel, CSV, txt, Avro, Parquet, ORC, EBCDIC (mainframe), unstructured files with metadata, including audio, video and visual files
JasperSoft	Amazon Web Services, Microsoft Azure HDInsights, Cloudera, IBM PureData System for Hadoop, MapR, SAP HANA, Terada, vertica	Marketo, Salesforce, NetSuite, MS CRM, SAP, Sugar CRM, Microsoft Sage X3, CentricCRM	-	MaxDB, Microsoft OLE-DB, Microsoft SQL Server, MySQL, Netezza, Oracle, ParAccel, PostgresSQL, PostgresPlus, SAS, SQLite, HSQLDB	Cassandra, MongoDB, AWS S3, Google Storage	pdf, xls, CVS, XML, RTF, ODF, text, HTML and SWF

Table 2.3: Open-source ETL tools, based on [65], [67], [43] and [57].

2.5 Data Visualisation and Reporting

Data Visualisation is the process of displaying data after analysis pictorially or graphically to understand complex data better and make better decisions. As so, analysts can use data Visualisation to acquire a better grasp of the data and gain more economically viable insights [61].

Because this work is intended to produce a Big data-driven Web application, it is considered that data Visualisation tools which integrate with programming languages are the best suited, primarily JavaScript, which is the most widely used programming language on the internet [70].

Table 2.4 presents JavaScript based frameworks for Data Visualisation.

Tool	**Programming Language**	**Characteristics**
D3.js [11]	JavaScript	D3.js is a JavaScript library for document manipulation with powerful visualization components and a data-driven approach to DOM manipulation that gives the developer a wide range of functionality.
Chart.js[12]	JavaScript	Chart.js is a open-source JavaScript data visualization framework that supports eight different chart types: bar, line, area, pie (doughnut), bubble, radar, polar, and scatter.
React	JavaScript	React is a user interface library written in JavaScript. Despite not being a pure data driven framework it is possible to integrate it with libraries such as nivo[13], Materia-UI[14] D3.js, Chart.js and others.

Table 2.4: Data Visualisation JavaScript frameworks.

3

Architecture and Methodology

As previously described, there are currently many frameworks, tools, and technologies to deal with the implications of Big data and the universe that surrounds it. This chapter reflects on the previously discussed themes and presents a technological stack to implement a solution.

A description of the system requirements to develop the proposed system is made as is an architectural overview of the system.

It illustrates the expected interaction flow between the end-user and the application, which directly correlates with the system developed and the functionality available.

Lastly, it presents the methodologies used in the integration of data sources and predictive models.

3.1 Technology Stack

After analysing and studying possible solutions and different approaches in chapter 2, the need to choose between a wide range of technologies is clear.

Regarding the infrastructure, it was chosen to use a Public Cloud Provider, namely Google Cloud Platform. This decision came from the widely previously discussed benefits of using Cloud computing for Big data. GCP was chosen as the provider to use because of Google's tendency for open-source development, an essential point regarding academic work, and its reputation. Furthermore, the products and services available on GCP fulfill every need of the project in hand.

In respect of Data Storage, it will be used GCP's solutions will be used, namely:

1. **Cloud SQL**: to store patient data in a tabular way (Relational Database Management System (RDBMS));
2. **Cloud Firestore**: documents with development elements, namely profiles, menus, databases, services, and users;
3. **Cloud Storage**: buckets of information that need to be accessed in runtime such as images, patient information, between others.

Respecting Data Management, especially the ETL process, the Pentaho Data Integration tool was selected. This tool is open-source and is one of the most widely used for ETL and Data pipelines in general. It presents a wide range of functionalities, ease of use, immense documentation, and an online community. Since the infrastructure of this work is under GCP, a Virtual Machine will be deployed and dedicated to implement, test and deploy the ETL processes.

About Data Visualisation and Reporting, in the context of a Web application, React was chosen. React.js is an open-source JavaScript (JS) library for creating user interfaces, and it can be integrated with other powerful frameworks such as D3.js or Chart.js. It is used to build massive web applications that can alter data without reloading the page, i.e., statelessly, making it ideal for a Big data-driven application where the users need to have the most recent and accurate data to make decisions. Furthermore, React's primary goal is to be quick, scalable, easy to use, and highly responsive. In order to deploy the application, the App Engine will be used, a fully managed, serverless platform for building and deploying large-scale online applications [22].

3.2 System Requirements

This section will present the critical requirements for the architecture to be developed. To ensure proper functionality and ease of use, the application should fulfill some requirements in the end-user context. Below is a brief description of the system requirements.

- **Identify and understand the data sources**

  To properly feed the Data Storage infrastructure, the data sources to integrate should be well known, meaning that it is necessary to identify its content, clearly understand it, and then arrange an efficient and secure way to collect the data from its source into the ETL process.

- **ETL process**

  The ETL process should be prepared to integrate the data sources efficiently, always assuring the integrity of the data. Since the complexity and volume of the data are considered, this process should also be as optimised as possible since the risk of having it up and running for extended periods is higher, which also increases the project's total cost.

  An important point regarding the ETL process is that every transformation[1] should be independent and wholly decoupled between each data source; thus, facilitating the future integration of different sources.

- **Data Storage and Management**

[1] The concept of transformations and jobs in ETL context will be deepened in other sections.

After the ETL process is concluded, it is necessary to address the data storage. The appropriate data storage platform should be chosen depending on the data characteristics and where it is needed. Also, the security and privacy of the data should be addressed throughout the whole process[2].

- **Provided services - Google Cloud Functions**

  The services provided by the application to the end-user will highly depend on the good functioning of the Google Cloud Functions (GCF) since those will perform the actions required to build the actual information and knowledge that feeds the application [3].

  Consequently, this process should be as accurate as possible and performed in real-time so that the end-user can get the most out of the tool.

- **User Interface**

  The Web application and the information and knowledge displayed in it should be accessed in an easy, understandable, and ergonomic manner by the end-user. Since this project focuses on a healthcare environment, the designated professionals must extract valuable insight as efficiently and effectively as possible.

3.3 Architecture Overview

As previously mentioned, this project aims to create an easy-to-use Web application capable of providing a great User Experience (UX) while integrating multiple sources of information in a healthcare environment, which inevitably leads the application to a Big data scenario due to the nature of the data. With that being said, the need to design and implement an architecture capable of handling such challenges arises.

In order to make this possible, this architecture is composed of several modules responsible for different system characteristics. An overview of the architecture and the modules that comprise it, the interactions between each other and the end-user can be seen in Figure 3.1.

The proposed architecture is a serverless approach to the several previously identified modules that need to be addressed to provide the end-user with the capability to extract actual knowledge from the data. As seen in Figure 3.1, the architecture can be divided into four essential layers: an ETL layer, an Data Storage and Management layer, an Data Processing and Analytics layer, and the User Interface (UI) layer, responsible for the interaction between the whole system and the end-user.

[2] Althought the security and privacy topics are not part of the scope of this thesis, those were taken into account when developing it.

[3] It is essential to mention that the core implementation (predictive models) of those services is out of the scope of this thesis, however, the required adaptations and optimisations will be discussed.

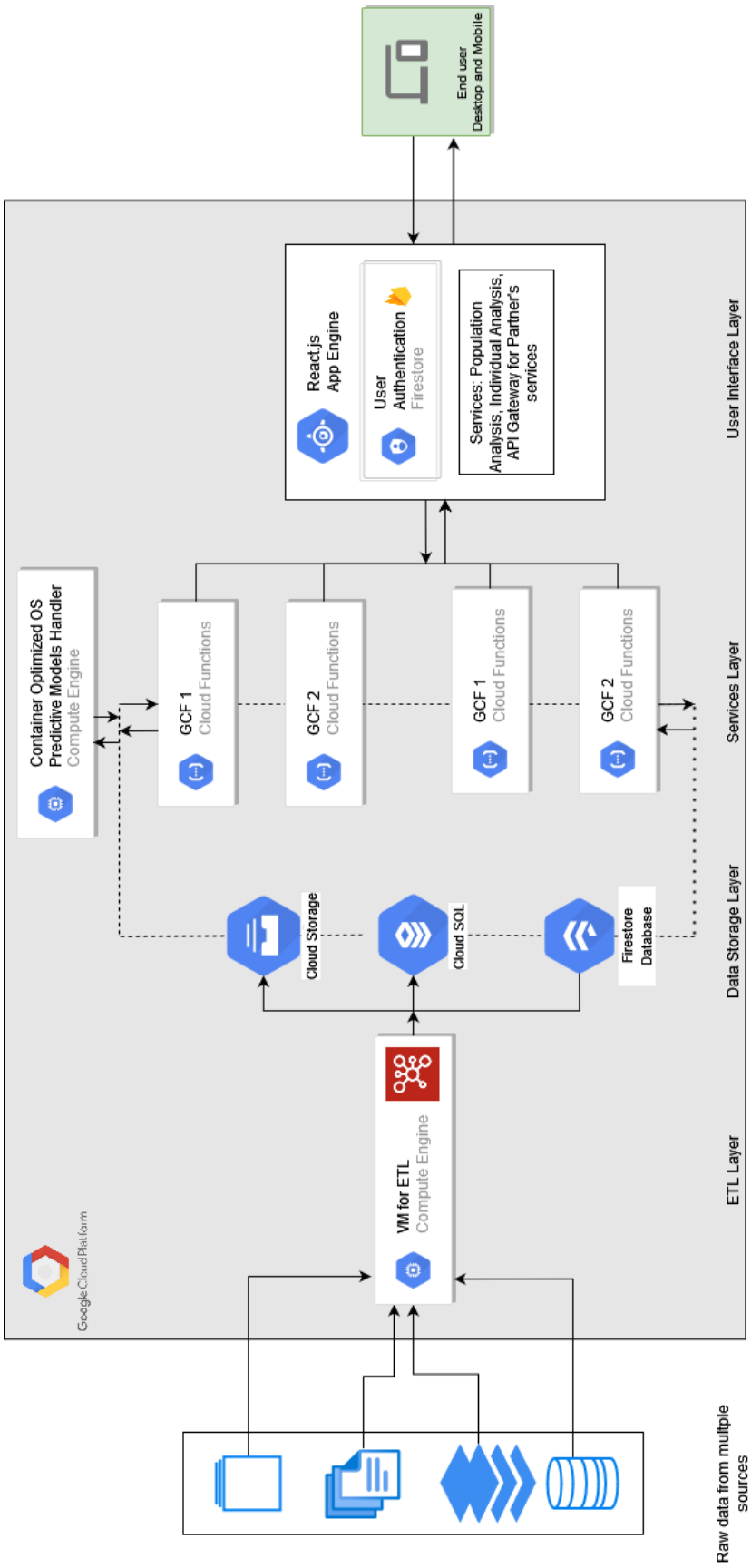


Figure 3.1: Big data-driven Architecture Overview.

The ETL is responsible for the integration of multiple data sources, performing adequate transformations to the data, and then loading the data into the appropriate storage structures. The Data Storage and Management works as a mediator between the ETL process and the Data Processing and Analytics since it will feed with clean and reliable data every forward processing. It is also important to mention that this module plays a crucial role in the whole system since its availability, speed, and accuracy will highly affect the UX and, as a consequence, the fact that the application is a valuable tool in a real-life scenario. Thus, the excellent choice of Data Storage and Management should be addressed carefully.

The Data Processing and Analytics will be responsible for providing ready to analyse data as a knowledge discovery tool; this module applies a series of algorithms and predictive models to the data and makes it available to consume as a service.

Finally, the UI between the system and the end-user; this module serves to provide a graphical user interface via a web page, which any device can access with an internet connection since the application should be responsive.

3.4 End-user Interaction

The user interaction with the application is limited by the developed graphical interface and the requests one performs. As a result, it is critical to make sure that these interactions are simple and convenient for the user. Figure 3.2 depicts the system response to a sequence of interactions to analyse data.

From the user's point of view, the first interaction with the system is authentication. The only way to register new users is by registering them in the Firebase admin console. This process is conducted exclusively by authorised staff, and then the user/password pair is addressed to the designated user to access the application. Also, when registering the user, it should be configured in the Firebase admin console, which type of user it is and which databases it can access and query. This process will be discussed in other sections.

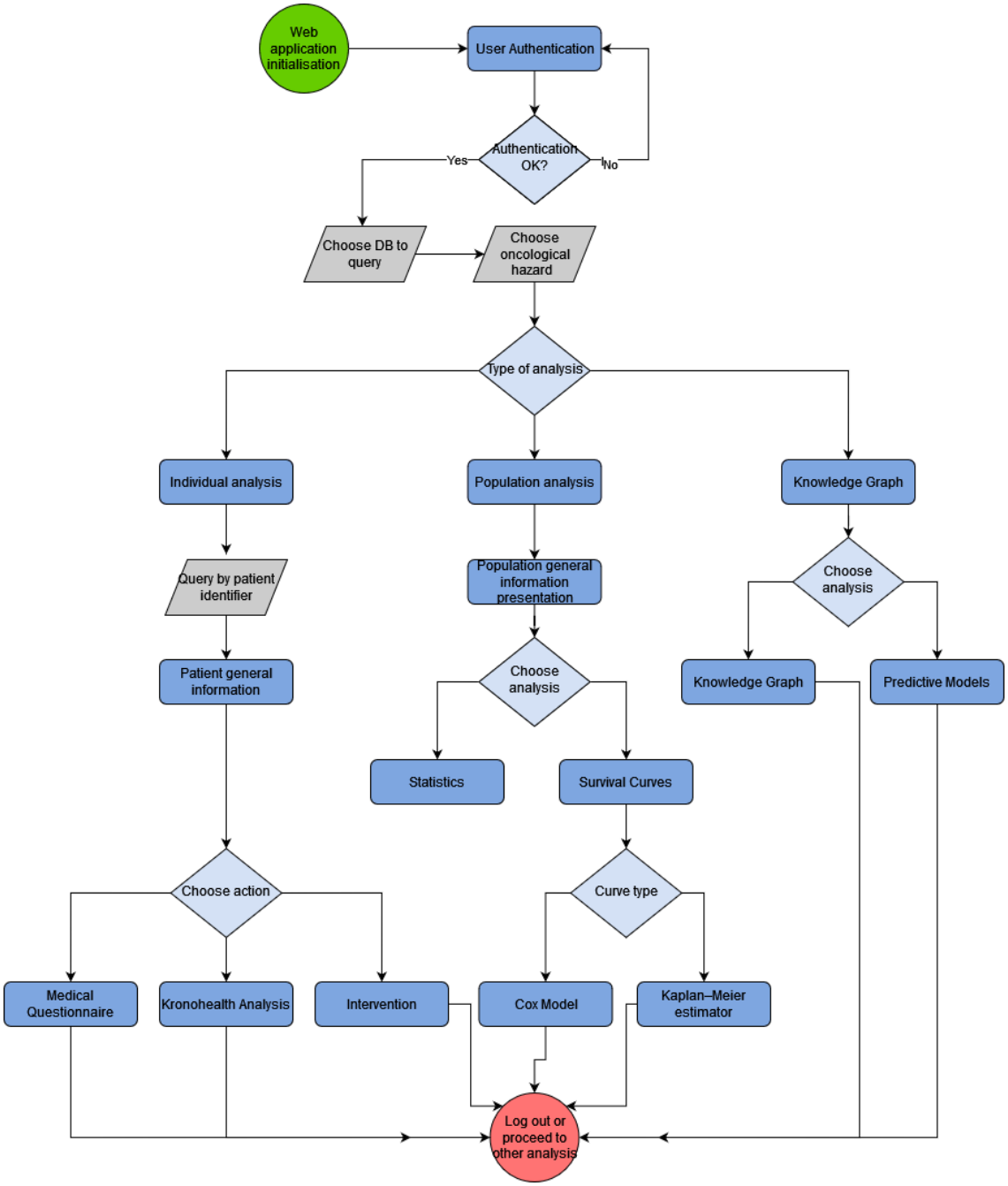


Figure 3.2: User interaction flowchart.

Since the authentication is granted, the user should choose the database that desires to be analysed, between the ones that they have access. It should also be mentioned that the user has no information about which other databases are available due to privacy questions. The user can choose between the available oncological hazard, directly affecting every further interaction with the system. However, it can be changed anytime and multiple times during the same session, allowing the user to perform several queries and analyses across different hazards. Once chosen the database and the hazard type, the user is presented with a menu to perform different kinds of analysis, being those:

1. **Individual analysis**: in this scenario, the user is asked to input a patient ID [4] and is presented with a panel with general information about the patient. From there, the user can choose different interactions with the system, which will be further addressed.

2. **Population analysis**: in this scenario, it can be chosen between two different approaches:

    a) Survival curves:

        i. Cox model

        ii. Kaplan-Meier estimator

    b) Descriptive Analysis:

        i. Statistics

3. **Knowledge graph analysis**: this submenu aims to integrate other functionalities that are outside of the ecosystem developed in the scope of this project, namely a drugs interaction API and predictive models. Those functionalities were created by other entities that collaborate in the Clarify project. Thus, its integration was done in thigh communication with the respective entities to preserve functionality and usability. It is also important to mention that those are not fed by the data structures previously mentioned.

3.5 Data Sources Integration Process

The process carried out to integrate the multiple data sources used in the project was iterative, accompanied by the Clarify project partners responsible for each source. Figure 3.3 intends to describe the data's process from its origin until its integration in the project's ecosystem.

[4] This information is highly confidential, and only authorised hospital staff have access to the suitable correspondence

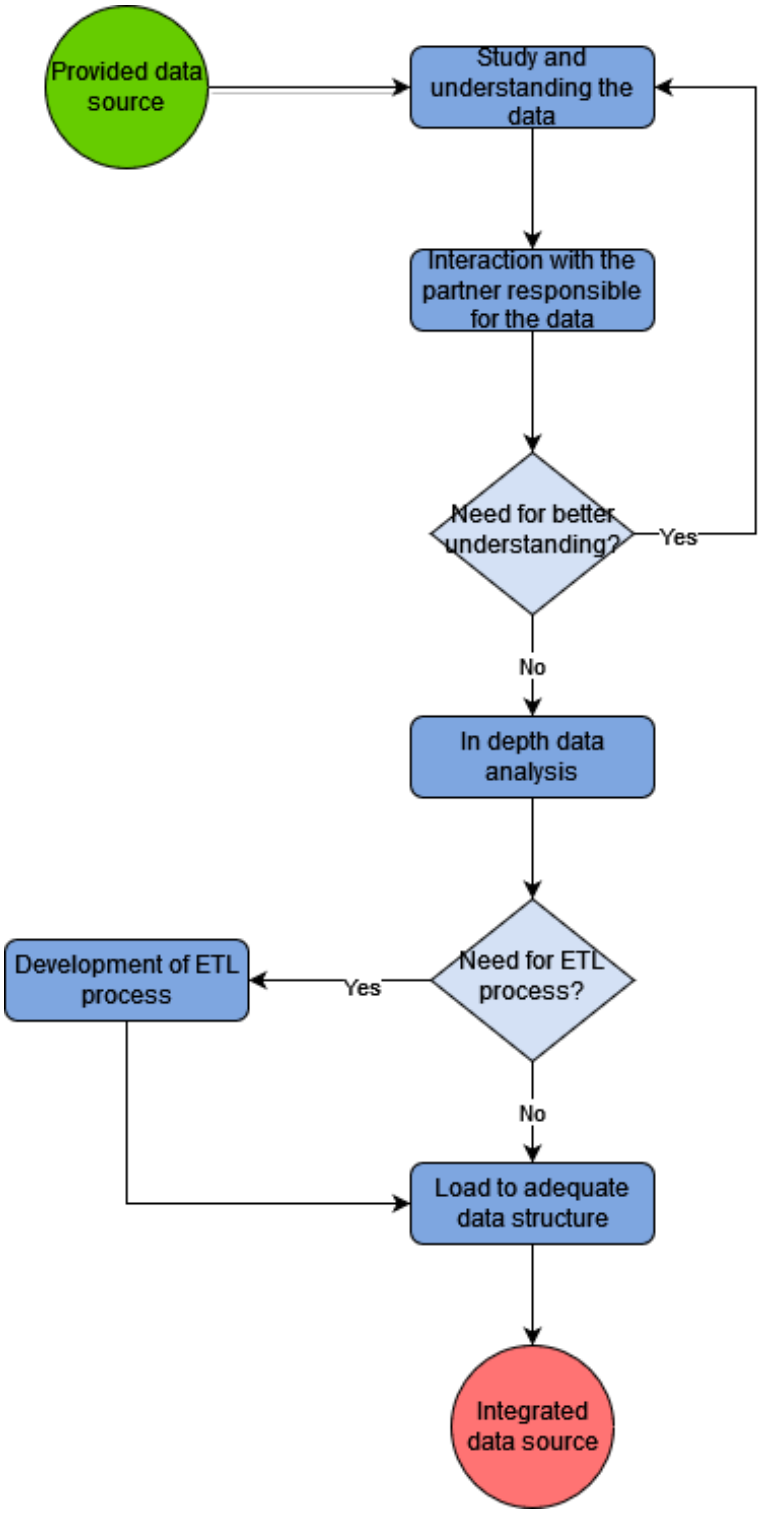


Figure 3.3: Data source integration process flowchart.

For choosing the adequate data structures to load the different data sources, it was necessary to do a previous study of the provided datasets to understand the context and its framework. Although this previous study was made, the necessity for a better understanding eventually emerged. It was the reason that an iterative process was built with the entities that provided the data. Hence, it was possible to completely understand the data, interpret it, and its constraints.

It is essential to mention that it imposes an increased difficulty since this work is inserted in a particular domain knowledge area.

For that same reason, without the required knowledge about the data, it was easy to run into an ineffective ETL process or load the data into inadequate data stuctures, which would affect the results built by the models in the subsequent phases.

After the interaction with the responsible entities is done and if it is possible to understand the data comfortably, the conditions to proceed are reunited. Then emerges the necessity to decide whether the data needs further processing and thus the development of an ETL process or it is ready to be loaded into a data structure.

3.6 Models Integration Process

The process to integrate the models was similar to the one described in the previous section. Since the provided models were already written in a programming language supported by the GCF, it was only necessary to perform some adaptations to use them. Those adaptations will be discussed later in subsequent sections. Figure 3.4 describes the process to put together the implementation of such models.

The process carried out focused mainly on ensuring that the integrity of the model was maintained. Although some adaptations were made to have the models available in the project ecosystem, it was fundamental to keep the results obtained by the entities responsible for developing the models. This process even brought significant advantages to the project since it was possible to detect eventual flaws and improve the quality and efficiency of the code used to build the models.

Since this project is tightly related to the usability of the web application and general UX, it is crucial to have the models available for interpretation as fast as possible, preferably providing the user with an almost real-time experience. This is, although the models are run in each request, its results should be presented in a few seconds to ensure usability. Since the development of the models was made in a scientific context, the performance and execution time was not the main topics of discussion. Still, that question had to be addressed in this scenario. Thus, a series of optimisations were implemented to accomplish this, those will be discussed in 4.3.2.

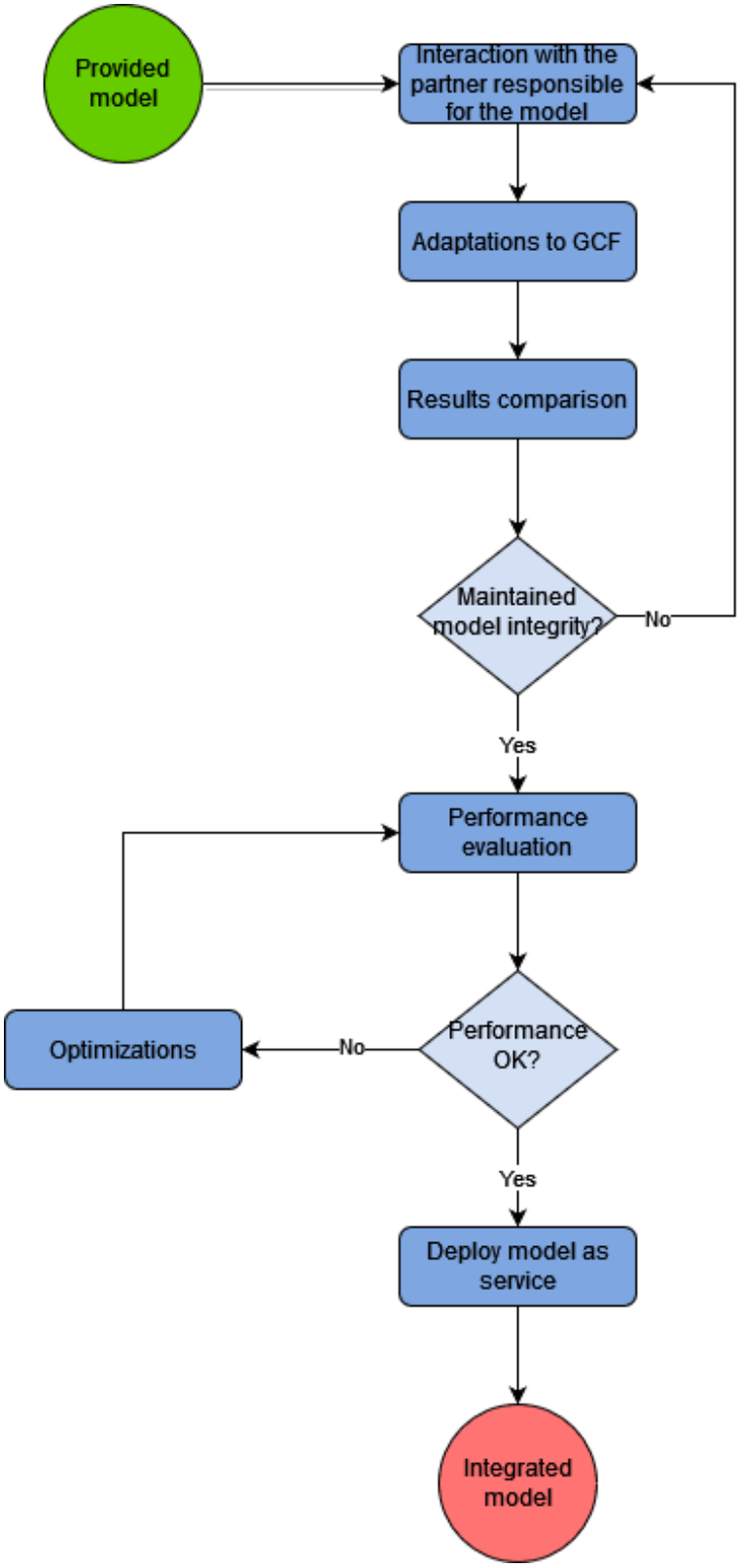


Figure 3.4: Models integration process flowchart.

4

Implementation

The implementation of the system previously mentioned and detailed in the preceding sections is described in this chapter.

Section 4.1 discusses the infrastructure implementation and the deployment of the necessary resources and services inside Google Cloud Platform to answer the project requirements.

Section 4.2 details the data sources used and the process of loading them to the project ecosystem and making them available for use in the project.

Section 4.3 presents the developed Google Cloud Functions and explains each service's goal and the type of communication used; it also discusses the implementation of optimisations to the developed services.

Section 4.4 details the developed User Interface and connected the dots between the UI itself and the developed services.

As mentioned in section 3.1, the infrastructure of the project is based on the Google Cloud Platform. This subsection discusses the core implementation of each module used in the project from the GCP ecosystem.

4.1 Infrastructure

The used services in the GCP ecosystem were:

1. App Engine
2. Compute Engine
3. Storage Buckets
4. Cloud SQL
5. Google Cloud Functions
6. Firebase

4.1.1 Google Cloud Platform Overview

Google Cloud Platform is a public cloud provider that offers various internet-based services related to hardware or software. Those services aim to hasten an organisation's digital transformation [58].

GCP is composed of physical assets like computers and hard disk drives, as well as virtual resources such as Virtual Machine (VM)s, all of which are housed in Google's data centers across the world. Each data center is part of an area. Asia, Australia, Europe, North America, and South America are all available as regions. Each area consists of zones that are separated from one another inside the region.

This resource distribution has various advantages, including redundancy in the event of a failure and reduced latency by placing resources closer to customers [31].

Any Google Cloud resources allocated must be part of a project. A project can be thought of as an organising entity. The settings, permissions, and other metadata that characterise the applications are part of the project.

Within a single project, resources can readily collaborate, for example, by communicating through an internal network, as long as the regions-and-zones guidelines are followed [31]. Each Google Cloud project has the following:

1. A project name;
2. A project ID;
3. A project number, which Google Cloud delivers.

These identifiers are used in certain command lines, configurations, and API calls. Figure 4.1 shows the project name, its ID, and its project number.

Figure 4.1: Project information on GCP.

Google Cloud provides three ways to interact with the services and resources, which are:

1. **Google Cloud Console**: provides a graphical user interface for managing Google Cloud projects and resources via the web.

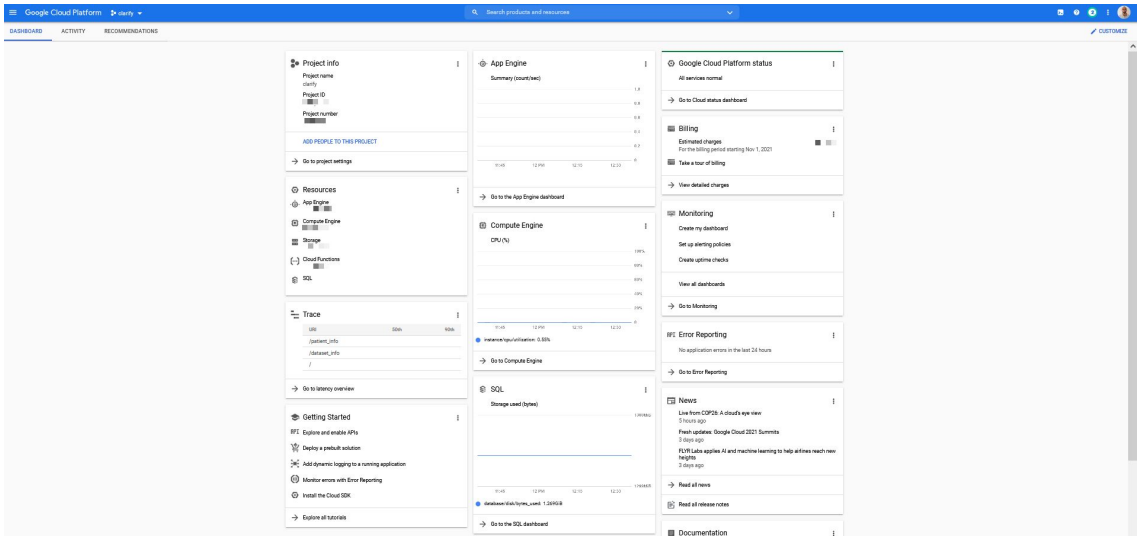


Figure 4.2: Google Cloud Console example.

2. **Command-line interface**: experienced users may prefer to use the command-line interface, which allows performing the same kind of actions and tasks as the graphical interface with more flexibility and precision.

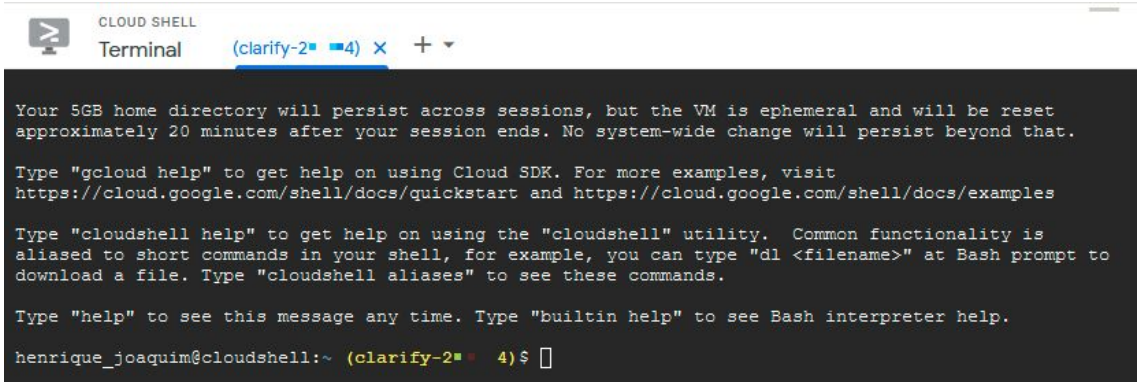


Figure 4.3: Cloud Shell (gcloud command-line) interface example.

3. **Client libraries**: the Cloud Software Development Kit (SDK) includes client libraries that enable easy creation and managing of resources. Those libraries expose APIs for two primary purposes:

   a) App APIs provide access to services;

   b) Admin APIs offer functionality for resource management.

4.1.2 Project Creation

To create a project in the Google Cloud Platform is necessary to have a Google domain account and associate it with a billing account. Thus, it is only required to access the official GCP website[1]. Figure 4.4 shows an example of the initial page after setting up the GCP account.

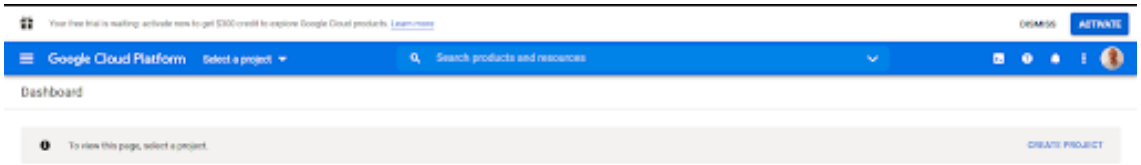


Figure 4.4: Initial page of GCP after setting up an account.

As it is possible to observe in Figure 4.4, the web page prompts the user to "Create a Project".

After clicking the button, the user is asked for a "Project Name", and the project initialisation is done, and all the GCP products are available to use.

4.1.3 App Engine

As previously discussed, the development of the UI will be done using React and recurring to the GCP's dynamic website hosting serverless solution App Engine. Thus, under the previously created GCP project, an App Engine Application should be made.

To deploy the application to App Engine, running it in the local development environment is necessary. Thus, Listing 4.1 presents the Cloud Shell local installation instructions that allow deploying a React application in localhost.

```
# Clone ARc template
$ git clone -b master https://github.com/diegohaz/arc.git my-app
$ cd my-app

# Creating a new repository
$ rm -rf .git
$ git init

# Install Dependencies
$ npm install

# Run the App
$ npm run dev

# App initialised on http://localhost:3000
```

Listing 4.1: Deploy application on localhost.

[1] console.cloud.google.com

It is essential to mention that the cloned template application is an Open Source GitHub repository[2] that focuses on ARc (Atomic React[3]), which is a React starter kit based on the Atomic Design methodology [39]. It will be used as the foundation for further development, which will be addressed in the UI section.

Since the application is running locally, it is ready to deploy to the App Engine, i.e., the production environment[4]. Listing 4.2 presents the Cloud Shell instructions used to achieve it, according to [33]. Although, before performing those sets of instructions, it is necessary to create an app.yaml Configuration File, according to [23].

```
# Transpile the source code into the dist folder
$ npm run build

# Deploy the application
$ gcloud app deploy

# Browse the application
$ gcloud app browse
```

Listing 4.2: Deploy application to App Engine.

Once the App Engine Application setup is done, the Web application is ready for further development. Whenever desired to launch an update to the application, it is only necessary to repeat the enumerated instructions described in Listing 4.2.

4.1.4 Compute Engine - Virtual Machine

The GCP's Compute Engine is an Infrastructure-as-a-Service that allows users to deploy virtual machines. In the scope of this project, a VM will be used to address the data processing and curation needs.

This section will address the deployment of such service, setting up a UI on the VM to allow a more user-friendly interaction, and finally, the installation of Pentaho ETL Tool, the previously discussed tool of choice.

4.1.4.1 Deployment

When creating a VM instance, it is recommended to do so through the Google Cloud Console. When searching for the same expressions, the user is prompted to a menu to create an instance. Figure 4.5 presents the produced menu[5].

Following that, the user is asked to configure the machine. The usage of the VM in the scope of this project will be especially data processing and preparation, which is not a

[2] https://github.com/diegohaz/arc

[3] https://bradfrost.com/blog/post/atomic-web-design/

[4] In order to deploy the application to the production environment it is necessary to compile it into the dist folder; "distribution"folder contains the compiled code/libraries [68].

[5] It is fascinating to observe that almost every GCP resource creation menu is accompanied with a quick-start menu that presents an easy-to-understand step-by-step guide to less experienced users.

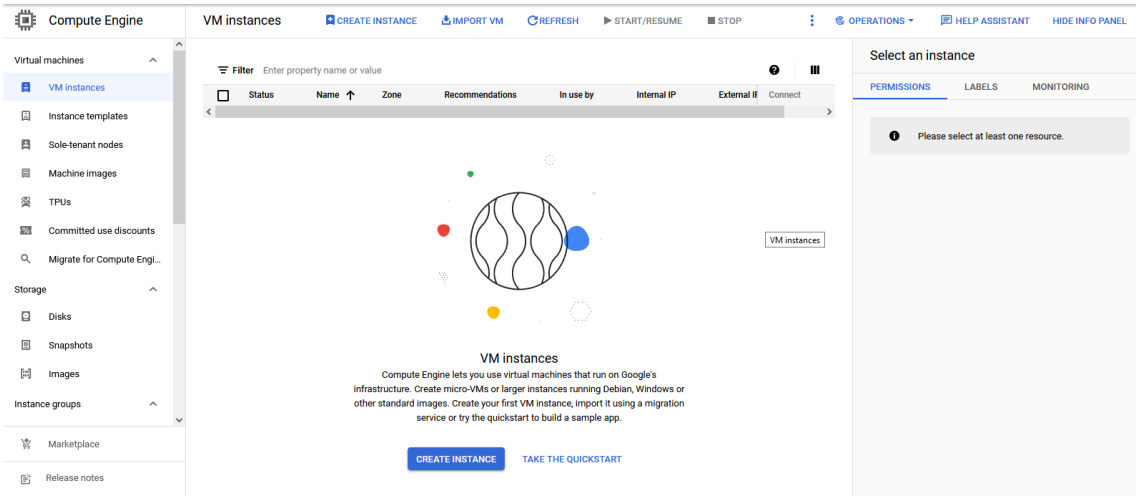


Figure 4.5: Create instance menu.

real-time task. Meaning that the medical data will be made available in periods separated by N days. It means that the VM can potentially be several days without being used.

When configuring the machine, it is intended to choose options that allow performing the necessary data processing and is cost-effective, considering the project needs.

With that in mind, it was chosen to configure a machine with standard GCP recommended characteristics, as follow:

1. **Region and Zone**: europe-west[6]
2. **Machine family**: general purpose[7]
3. **Series**: E2
4. **Machine type**: e2-medium (2v CPU, 4 GB memory)
5. **Boot Disk**: Ubuntu 18.04 LTS (10 GB)
6. **Firewall**: all incoming traffic from outside the Virtual Private Cloud (VPC) is blocked

After that, the machine is ready to use and can be accessed via Secure Shell (SSH) as seen in Figure 4.6.

Figure 4.7 presents the interface displayed when accessing the VM through the SSH.

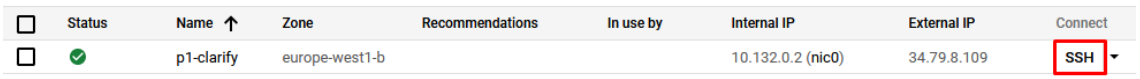


Figure 4.6: Configured VM ready for usage.

[6]The chosen data center should be as close to the users as possible to minimise latency.
[7]Machine types for common workloads, optimised for cost and flexibility [30].

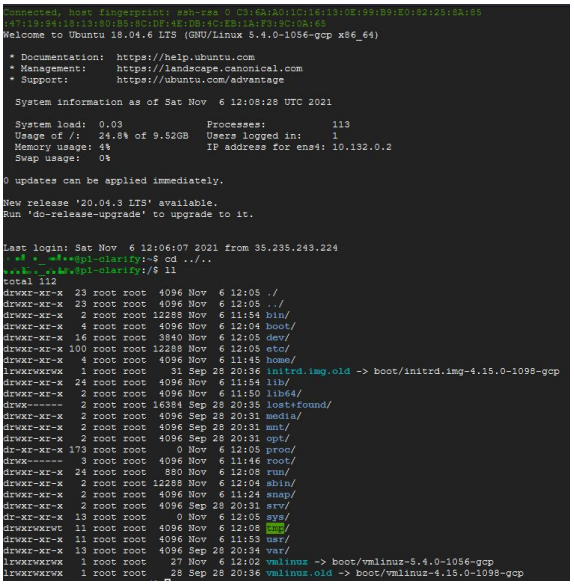


Figure 4.7: Configured VM SSH interface with typical Linux filesystem.

4.1.4.2 Access Virtual Machine through Virtual Network Computing (VNC)

Considering that the data needs to go through an ETL process and the VM will be the infrastructure that allows it to do so, it is necessary to access it in a user-friendly way. For that reason, the necessity to set up a UI to access the VM and develop the ETL process arise[8]. Listing 4.3 presents the necessary instructions to set up the VNC server on the VM.

```
# Update source list and install tightVNC (https://www.tightvnc.com/)
$ sudo apt-get update
$ sudo apt-get install tightvncserver

# Installing Xfce\footnote{https://www.xfce.org/} Desktop Environment */
$ sudo apt-get install xfce4 xfce4-goodies

# Making the desktop environment accessible via VNC
$ vncserver

# Check if the VNC server is running on port 5901 with netcat
$ nc localhost 5901

# 'RFB 003.008' should be returned
```

Listing 4.3: Setup VNC.

Since the VNC server is configured, it is necessary to create a new firewall rule to allow a designated IP address to access it; this can be achieved by navigating to the Networking - Firewall menu on the Google Cloud Console. Figure 4.8 presents the process.

[8]Pentaho ETL Tool is a drag and drop development tool, and for that reason, it is not possible to create processes only using the command line interface

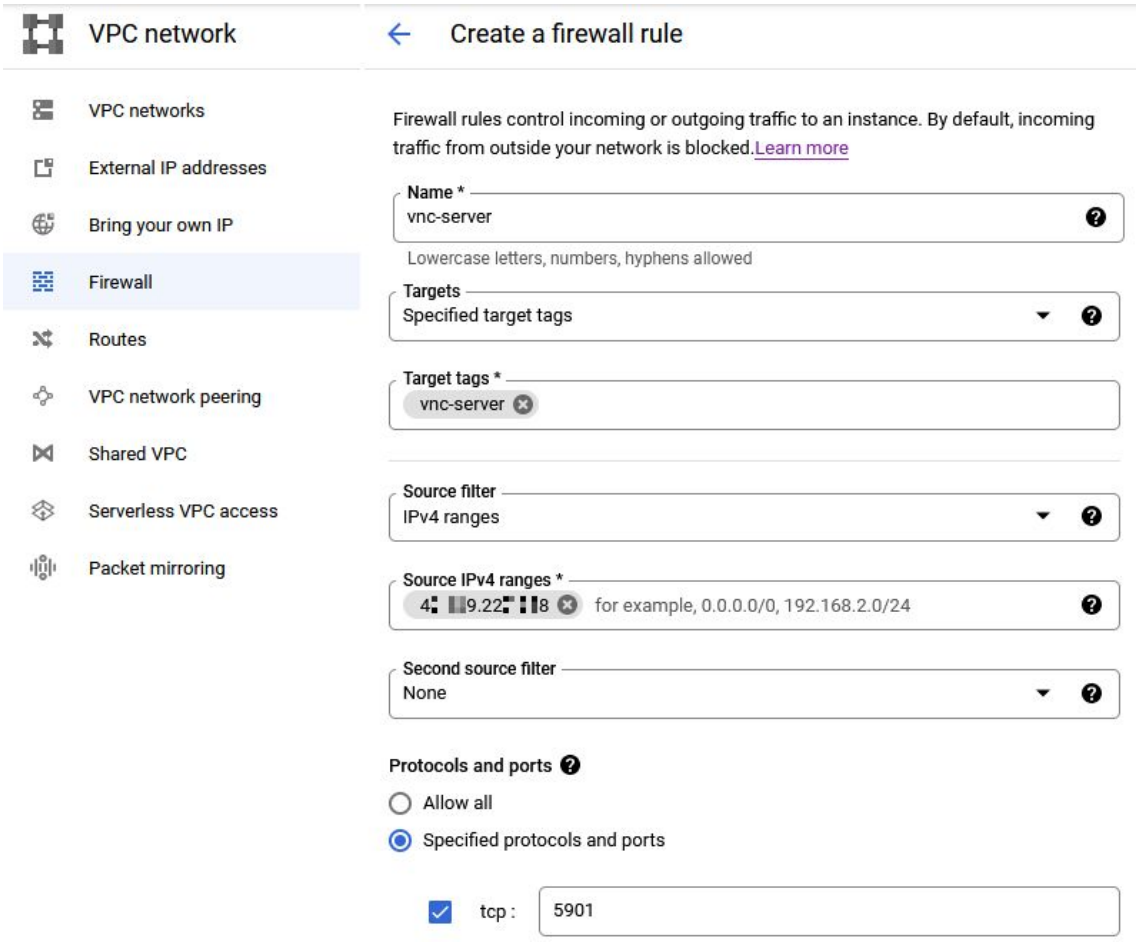


Figure 4.8: Setting up a new firewall rule.

After the firewall rule is created, it is only necessary to install a VNC client on the local machine, such as VNC Viewer [9] to access the VM. Figure 4.9 displays the result after connecting to the VM through the local machine.

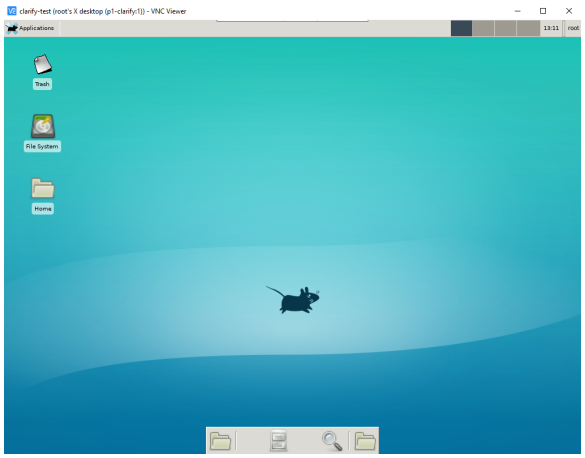


Figure 4.9: VM UI on VNC Viewer.

[9] https://www.realvnc.com

4.1.4.3 Pentaho ETL Tool Installation

In order to install the Pentaho ETL Tool, the responsible framework for the ETL process, it is necessary to install Java 1.8 and download the compressed file. Listing 4.4 illustrates the steps required to do so. Figure 4.10 illustrates the expected result.

```
# Installing Java 1.8
$ sudo apt install openjdk-8-jdk

# Define Java 1.8 as default Java version
sudo update-alternatives --config java
sudo apt install default-jre

# Download Pentaho ETL
$ wget https://sourceforge.net/projects/pentaho/

# Unzip downloaded file
$ unzip download

# Navigate to the unzipped folder and run Pentaho ETL
$ cd data-integration
$ sh spoon.sh
```

Listing 4.4: Setup VNC.

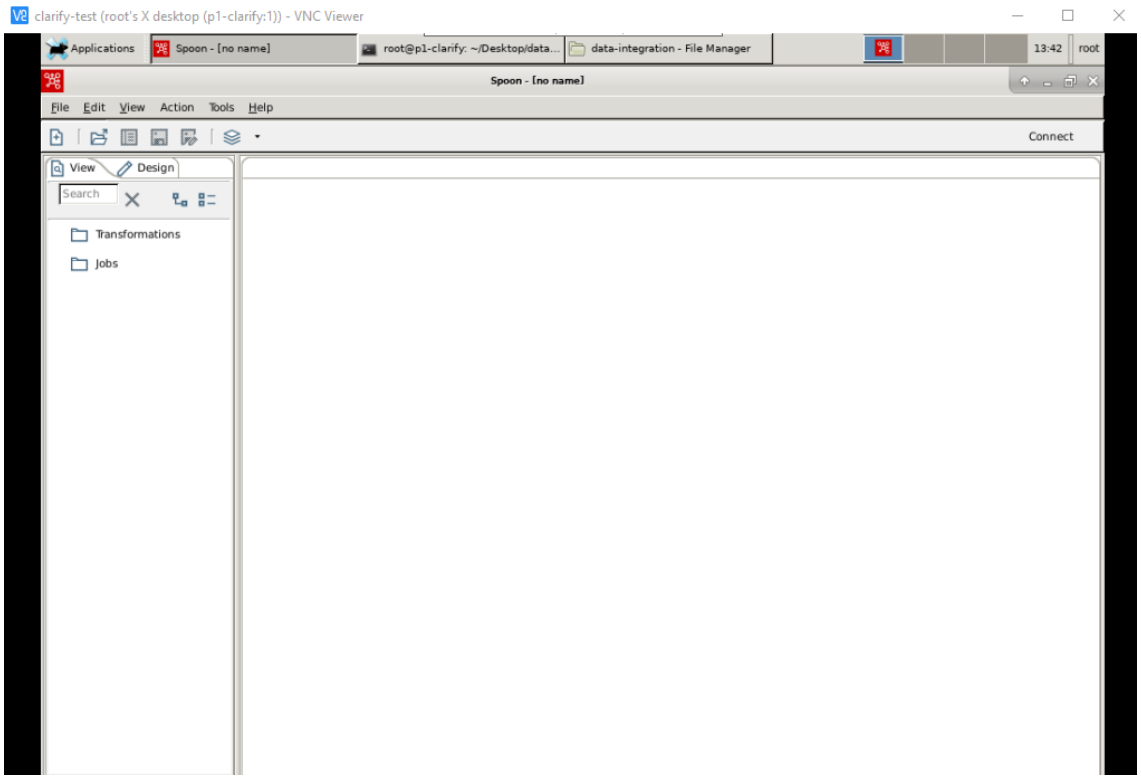


Figure 4.10: VM UI on VNC Viewer.

4.1.5 Cloud Storage

Cloud Storage is a GCP service for object storage. An object can be thought of as an immutable piece of data consisting of a file of any format [36]. Those objects are stored inside containers called buckets. Listing 4.5 presents the necessary instructions to create a bucket in the project and display its metadata.

The upload and handling of objects inside Cloud Storage and its buckets will be addressed in further sections.

```
# Create bucket
$ gsutil mb gs://BUCKET_NAME

# Display bucket metadata
$ gsutil ls -L -b gs://BUCKET_NAME
```

Listing 4.5: Creating a Cloud Storage bucket.

4.1.6 Cloud SQL

In order to store tabular data, it was decided to use a RDBMS. Thus, the adequate GCP service is the Cloud SQL. This service allows deployment of a database in the most common glssql versions, namely MySQL, Postgres, or SQL Server [27].

In order to deploy such a service, it is necessary to search for that same expression in the Google Cloud Console and then select the creation of an instance in the prompted menu. Figure 4.11 presents the available options to create the instance.

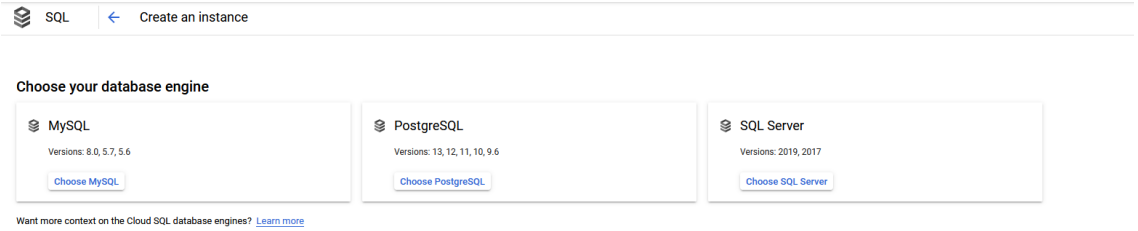


Figure 4.11: Available SQL versions to create a Cloud SQL instance.

In the scope of this project, it was decided to use the MySQL version. According to [53], it is the most widely used version and, as a consequence, it is expected that it is also the one with better documentation and support.

For the configuration, it maintained the default recommended options. However, it is possible to change the default anytime, for example, if more storage is necessary or n increased throughput.

Figure 4.12 presents the final configuration of the created instance.

MySQL instance

Region	europe-west1 (Belgium)
DB version	MySQL 5.7
vCPUs	4 vCPU
Memory	26 GB
Storage	100 GB
Network throughput (MB/s)	1,000 of 2,000
Disk throughput (MB/s)	Read: 48.0 of 240.0 Write: 48.0 of 240.0
IOPS	Read: 3,000 of 15,000 Write: 3,000 of 15,000
Connections	Public IP
Backup	Automated
Availability	Multiple zones (highly available)
Point-in-time recovery	Enabled

Figure 4.12: Created MySQL instance.

For the database connection, there are two options:

1. **Internal**: only VPC, that is, only accessible inside the project resources;

2. **External**: internet-accessible Internet Protocol (IP) address.

To provide a more straightforward development, it was chosen to have a mixed solution. The instance is always available on the VPC, and it was defined to be accessible to a list of authorised networks[10].

With that, it is possible to access the database with an authorised IP address in a MySQL[11] local installation. When it comes to authentication, built-in database authentication is used[12].

4.1.7 Google Cloud Functions

This section discusses the basic implementation and deployment of a Google Cloud Function. In further sections, it will be discussed the actual implementation of each GCF used in this project.

[10] A list of IP addresses allowed to connect.
[11] https://www.mysql.com/
[12] Log in with a username/password set in the database engine.

To deploy a Cloud Function, one should navigate to the designated menu on the Google Cloud Console and select the Create Function option. When creating a function, the user is asked for configurations, namely:

1. Function name;
2. Region;
3. Trigger type;
4. Authentication;
5. Memory allocated;
6. Maximum and minimum number of instances;
7. Ingress traffic.

In the scope of this project, the core configuration for every Cloud Function was the same. Figure 4.13 shows an example of a configuration.

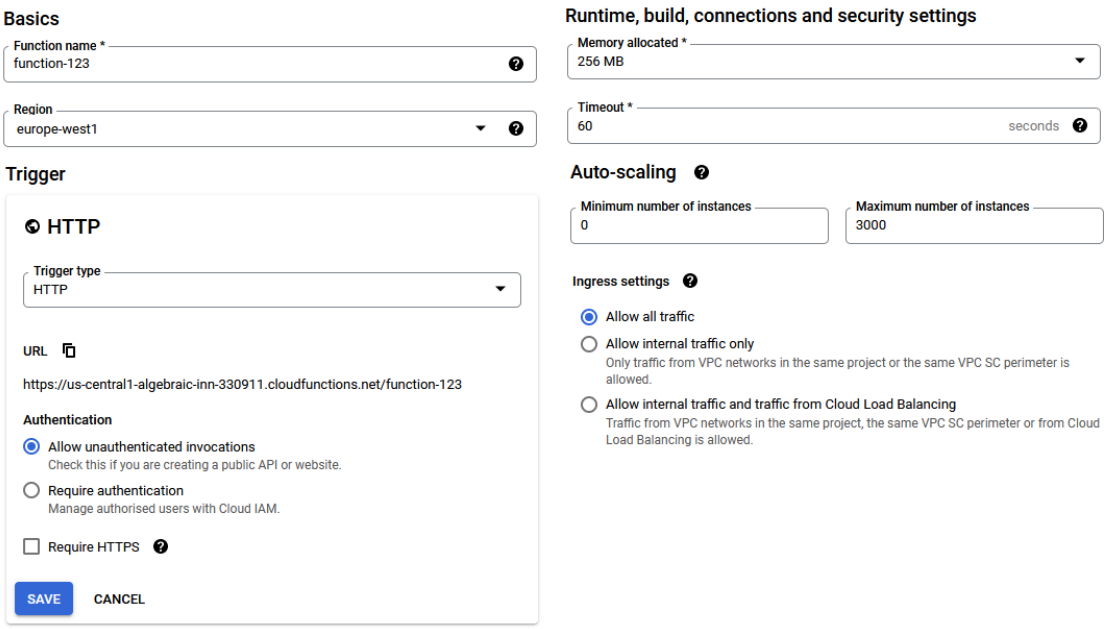


Figure 4.13: Example of a Cloud Function Configuration.

It is at the user's choice regarding the function's name, although it has to be unique by project and region. In the trigger section, it was chosen to have an HTTP trigger instead of an event-based trigger since it provides greater flexibility and generalisation, allowing the developer to think of the function as an API. This decision also relates to the fact that at the date of writing of this document, the invoked Cloud Functions are used to present data to the end-user by simple interactions in the UI and not by complex events that require further interaction with databases or other resources.

Regarding Authentication, the configuration of the Cloud Functions allows an unauthenticated invocation by default. This settlement was made to give the developer greater flexibility and allow personalisation regarding user access to resources. Each Cloud Function has its means of check authentication, particularly, check if the user is authenticated and if the user has permission to access the designated resource.

It was left with the default GCP values, memory allocation, timeout and auto-scaling, although it can be changed anytime if the necessity arises. Concerning the traffic ingress, all traffic will be allowed to ease the development process. However, in a production environment, it should be considered the option of only allowing internal traffic, i.e., the Cloud Functions would only accept invocations from the project's ecosystem, concretely, from the App Engine, which is the entry point for user's requests.

After the configuration process, the user is redirected to the development of the function itself. Figure 4.14 presents a basic example from the GCF documentation[13] and the development environment.

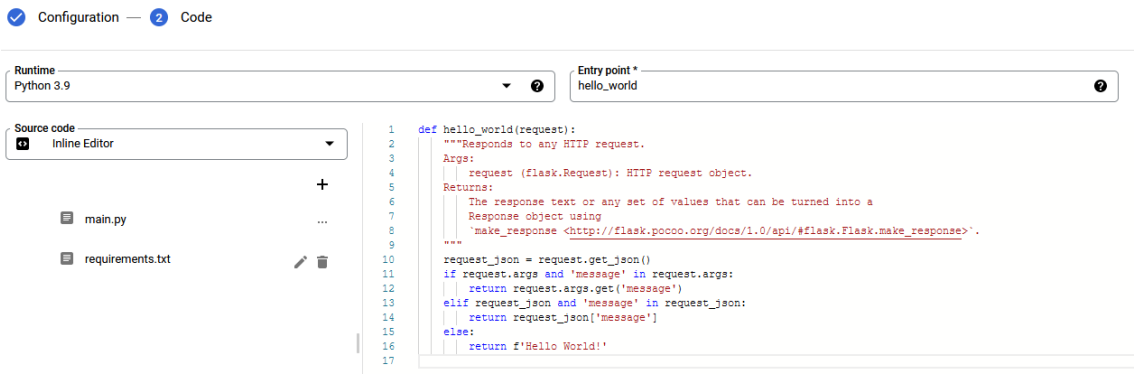


Figure 4.14: GCF development environment with a basic example.

GCP makes available a simple interface where the developer can set up a function. The user is asked for:

1. Runtime;
2. Entry point.

The runtime is essentially the programming language used to develop the Cloud Function. In the scope of this project, it was selected to use Python 3.9[14] in every Cloud Function. This decision came from the fact that the Cloud Functions required by the project highly rely on data science-related models, and Python is one of the most used frameworks in the field. Therefore, facilitating the development of such models by the project partners. The entry point is the method that will execute when invoking the Cloud Function, i.e., Cloud Function can have several methods defined in the "main.py," used inside the entry point method.

[13] https://cloud.google.com/functions/docs/quickstart-python
[14] https://www.python.org/

As it is possible to observe in Figure 4.14, there are two separated files:

1. "main.py": used for methods development;
2. "requirements.txt": used to specify dependencies.

It is essential to mention that the Cloud Functions menu allows the user to access a series of metrics such as invocation per second, execution time, memory usage and active instances, logging information, and a testing section. Also, the majority of the initial configuration can be changed on need, except the region.

To test the functionality of the deployed Cloud Function, it can be used with an API testing platform such as Postman[15]. Figure 4.15 shows an example of an invocation to the previously seen GCF example.

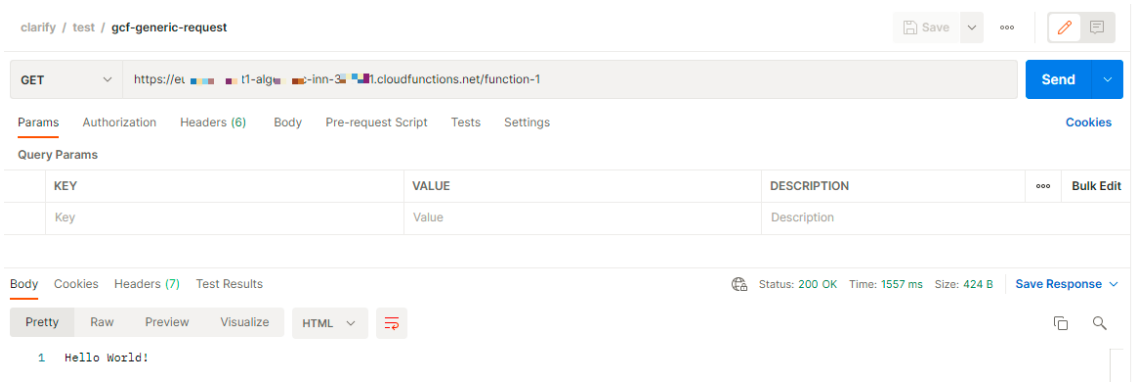


Figure 4.15: Example of GCF invocation through Postman.

Of note, the process of deploying a Cloud Function can also be done through the Cloud Shell. Listing 4.6 presents the necessary instructions to achieve it.

```
# Navigate to desired local directory
$ mkdir exampleCloudFunction
$ cd exampleCloudFunction
$ touch main.py
$touch requirements.txt

# ### Development process ###

#Deploy Cloud Function
$ gcloud functions deploy exampleCloudFunction --entry_point --runtime
    python39 --trigger-http --region europe-west1 --allow-unauthenticated
```

Listing 4.6: Deploy example Cloud Function.

As noticed, configuration parameters are not being used compared to the Google Cloud Console deployment. For those, it will be used the default values; although it is possible to configure every parameter from the Cloud Shell, as described in [54]. This solution can be particularly interesting if there is a need to use subversion software and

[15] https://www.postman.com/

empowers the developer to perform deployments from the local machine, which eases the development process.

4.1.8 Firebase

Firebase is Google's web and mobile development platform, and it can be used together with Google Cloud Platform to expand the functionality of an existing application [24].

According to [25], the Firebase Authentication allows multiple user authentication mechanisms. Including Google, Facebook, and Twitter. It also accommodates the largest number of users with the least amount of code. A GCP project is directly linked to the Firebase, which allows the developer to proceed to Firebase without an additional account or user configuration. Figure 4.16 shows the Firebase console menu after authentication and project selection.

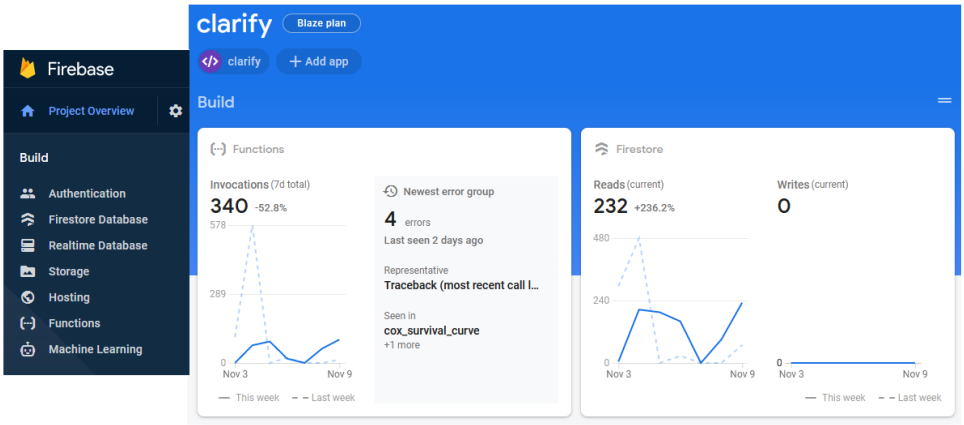


Figure 4.16: Firebase console.

As it is possible to observe, some of the Firebase services overlap with the previously discussed GCP services; this is because both the GCP and the Firebase can be used wholly decoupled from each other. Although used together for the same project, both present the same information regarding metrics about the application, leaving the metrics and logging observation to the developer's preference.

In this project, the goals of using the Firebase interface undergo three main reasons:

1. End-user authentication;
2. UI menus;
3. Rule definition.

The end-user authentication will be using the native provider Email/Password sign-in method. At the date of writing this document, the users can only be registered by the system administrator. Figure 4.17 presents the referred way enabled. Whereas the user registration can be made in the Users tab as presented in Figure 4.18.

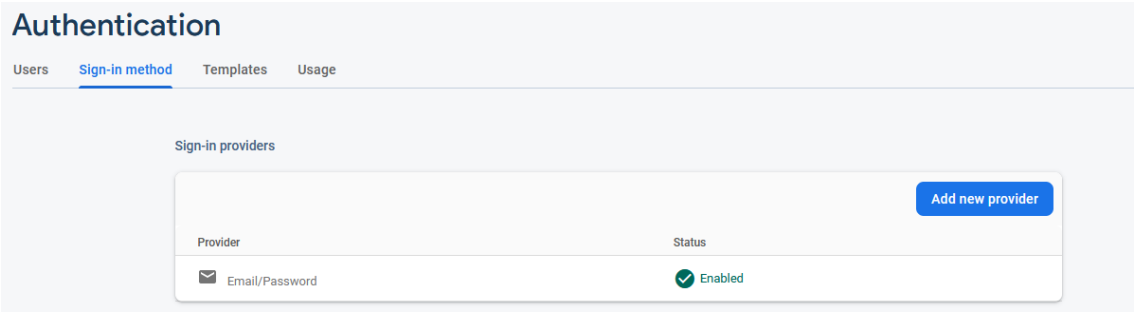


Figure 4.17: Authentication methods on Firebase.

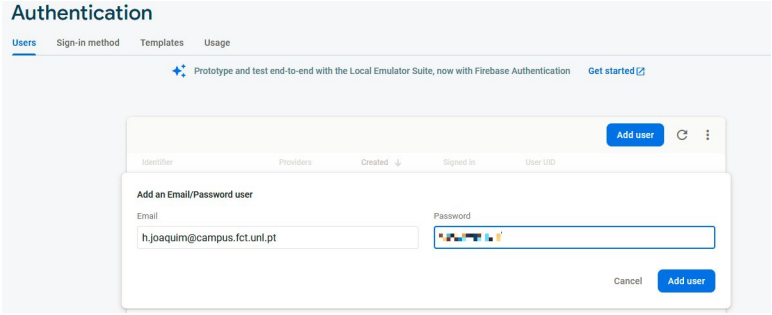


Figure 4.18: User registration on Firebase.

Instead of defining the UI menus in the coding development process, it was settled directly on Firebase. To perform this action, it was used the Cloud Firestore, a NoSQL, document-oriented database. This approach can provide a high-level abstraction of the application, which provides the developer with a higher degree of comprehension of the application and, consequently, makes it easier to make engineering decisions about integrating new functionalities. The Cloud Firestore is organised as Collections that contain Documents that, in turn, have its data. Figure 21 presents an example of that organisation in the Cloud Firestore interface.

This kind of representation can typically be thought of as a JavaScript Object Notation (JSON) object. As so, Listing 4.7 displays the same example presented in Figure 4.19 as a JSON object.

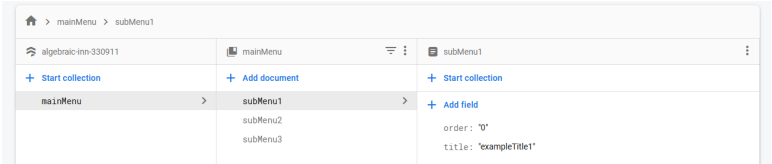


Figure 4.19: Example of a menu in Cloud Firestore.

```
{
    "mainMenu": {
        "subMenu1": {
            "order": 0,
            "title": "exampleTitle1"
        },
        "subMenu2": {
            "order": 1,
            "title": "exampleTitle2"
        },
        "subMenu3": {
            "order": 2,
            "title": "exampleTitle3"
        }
    }
}
```

Listing 4.7: Example of a menu organisation as a JSON object.

Regarding the rule definition, the goal is to provide the administrator with the ability to define rules quickly and efficiently. As an example, Figure 4.20 illustrates a list of hospitals and the definition of a user that can only access the designated hospitals. Listing 4.8 presents the same example defined as a JSON object.

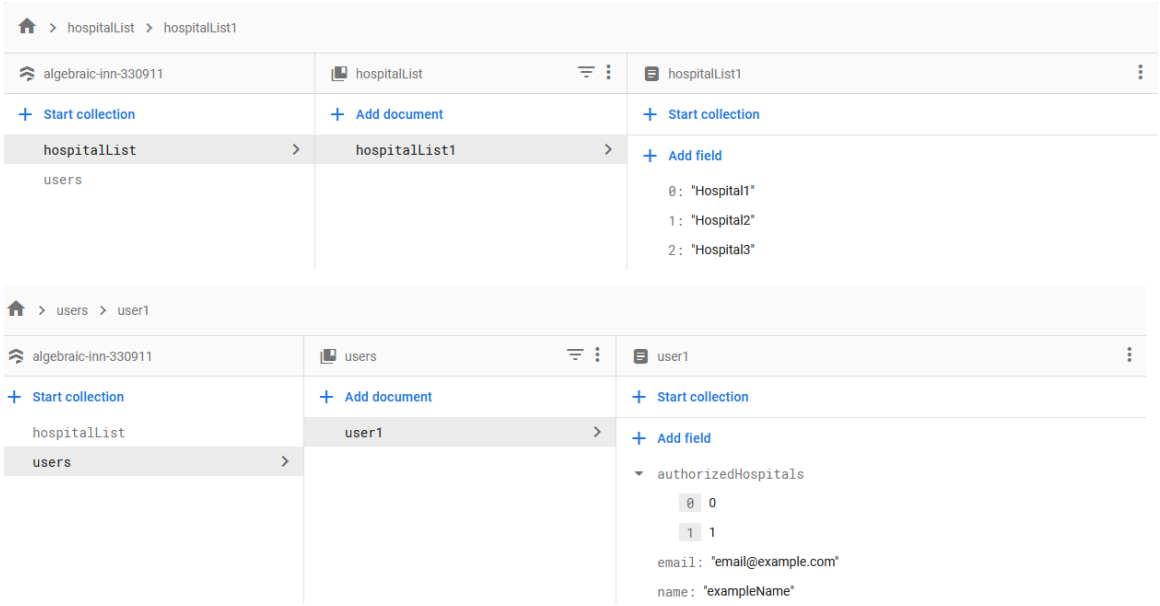


Figure 4.20: Rule definition example on Firebase.

```
# Hospital Listing
{
    "hospitalList": {
        "hospitalList1": {
            "0": "Hospital1",
            "1": "Hospital2",
            "2": "Hospital3"
        }
    }
}

# User definition with authorised hospital access definition
{
    "users": {
        "user1": {
            "authorizedHospitals": [0,1],
            "name": "exampleName",
            "email": "email@example.com"
        }
    }
}
```

Listing 4.8: Example of a rule definition as a JSON object.

4.2 Data Sources

In this section, the data sources used in the project will be presented. The focus will be on the analysis the amount of data, typology, characteristics, and brief analysis for each source. The necessity or not of developing an ETL process will be approached and, when needed, presented the development of that process. Finally, it discusses the most suitable data structures to load the data source in hand.

4.2.1 Population Data

The population data consists of a spreadsheet with several sheets provided by one of the project partners. Concretely, an hospital. Since the provided data is in the tabular format, it was decided to use Cloud SQL as the final data structure to store such data; at least while it does not hit a Big data paradigm.

The agreed typology of the spreadsheet consists of the following sheets:

1. Demographics;
2. Treatments;
3. Radiotherapy;
4. Surgery;
5. Progressions.

Every sheet has the same number of registers, each representing a patient. The provided dataset has approximately fifteen thousand patients, which translates in around 100 MB.

Although that amount of data is not near from a Big data paradigm, it is easy to understand that that same paradigm is easily reachable with the increase of hospitals and the inevitable increase of historical data, which will eventually support the computed models to build better ones.

Each sheet relates to the others by a unique identifier, i.e., the information about a patient is spread across the five tables.

Figure 4.22 presents the implemented database schema.

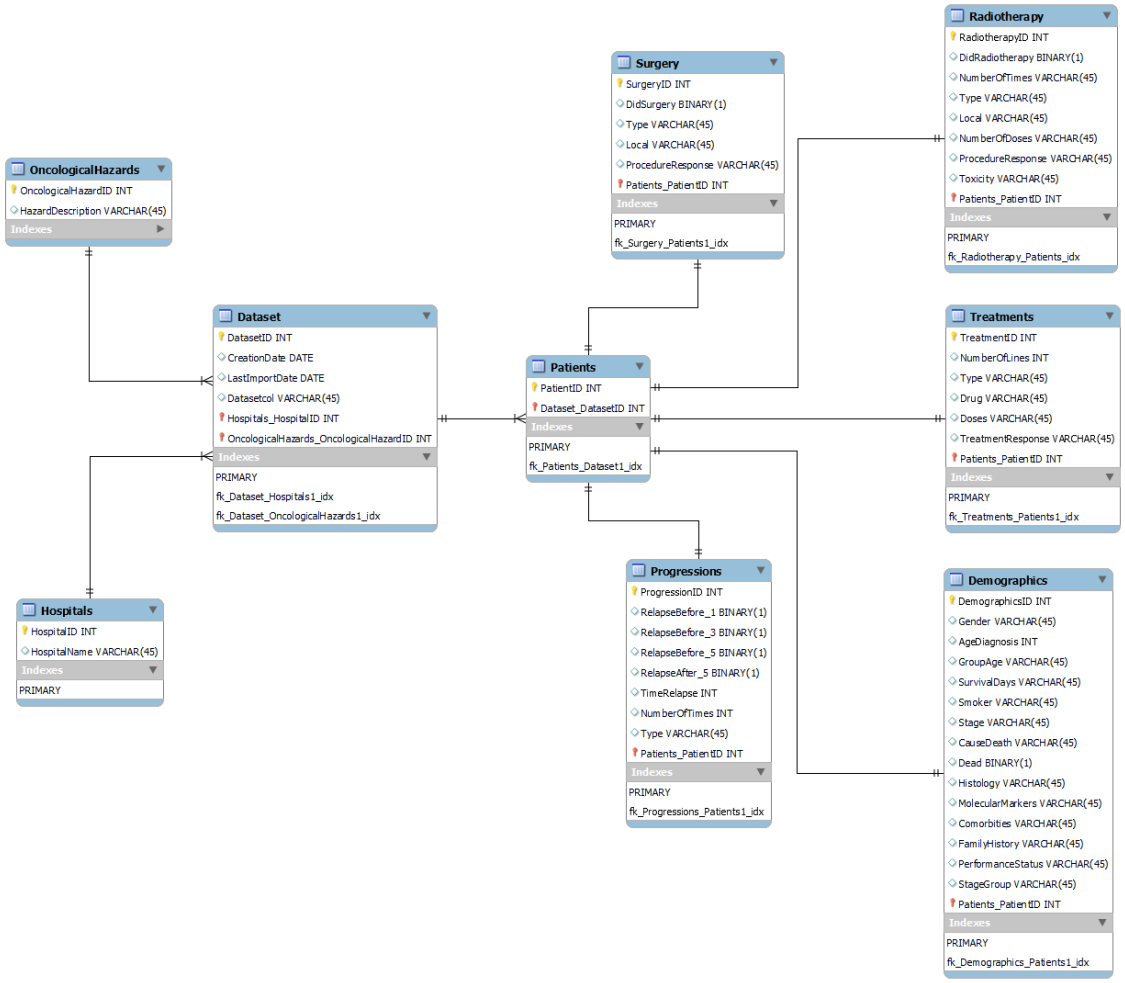


Figure 4.21: Database Schema Model on MySQL Workbench.

As it is possible to understand, the schema is centered in the Patient table. Each patient comes from a dataset, and a dataset is defined by the origin hospital and the type of oncological hazard.

The Hospitals, OncologicalHazards, and Dataset tables can be treated as tables that contain constant values, i.e., will not change in time; therefore, its content was manually introduced, querying the database.

As previously mentioned, the information about a patient is separated into five areas that translate into five tables:

1. **Demographics**: demographic information, i.e., information about statistical characteristics of a designated population;

2. **Progressions**: information about the patient's relapse and the progression of an oncological hazard;

3. **Radiotherapy**: data about the patient's radiotherapy treatment, it has done, dosage, treatment response, and toxicities;

4. **Surgery**: details concerning the surgery, if it has been done, number of times, type, and procedure response.

5. **Treatments**: particulars regarding the treatments done, such as the number of treatment lines, used drugs, and treatment response.

Of note, the provided spreadsheet contained more than a hundred variables each; to design a comprehensive, usable, and garbage-free database, the relevance of each variable was widely discussed with the medical staff and the project partners responsible for the data analysis models.

4.2.1.1 ETL Process

In order to load the data into the designed database, it was necessary to develop an ETL process, recurring to the previously selected ETL Tool, Pentaho. The primary goals of such a process, in this context, are:

1. **Extract**: the need to automate the extraction of data from a provided spreadsheet.

2. **Transform**: necessity to transform variables, perform mappings, clean missing values; this is especially important because the data analysis models are computed each time an end-user request is made; with this, the data that feed the Cloud Functions are entirely ready for use, i.e., no need for extra computation in order to prepare data; with this strategy, the application can give an answer in a shorter time, which significantly improves the UX.

3. **Load**: it is necessary to load the transformed data to the previously designed relational database.

It was developed an ETL pipeline consisting of a Job containing six Transformations, each of those transformations, responsible for a sheet and, consequently, a table. Figure 28 illustrates the developed Job.

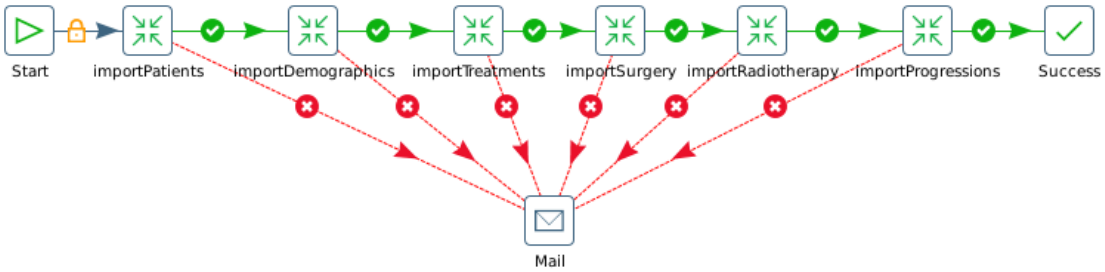


Figure 4.22: ETL Job in Pentaho.

As it is possible to observe, the developed Job contemplates all the transformations; in case of failure, an email is sent to a configured address[16]. The failure event occurs when the Pentaho Job aborts; that can happen if a file has the wrong format, in case of wrong variable types, fails to connect to the database, and other unexpected events.

As it is possible to verify in Annex I, the approach used to develop every transformation was similar. After accessing the database in order to verify the existence of a previously inserted record, a missing values validation recurring to "JavaScript Modified Value"[17] is done; the records with missing values are withdrawn from the process. Subsequently, a series of mapping is done, and, lastly, it is checked whether the record already exists in the database, and if that is the case, an update is made; otherwise, a new record is inserted.

4.2.2 Kronohealth files

The Kronohealth[18] files are provided by one of the project partners. Those files are reports generated by a wearable device that collect data about sleep habits and circadian rhythms during a designated period.

Oncological patients accompanied by the medical staff who collaborate in the project are asked if they want to participate in the study and use the Kronohealth wearable device; the patients who agree are then forwarded to Kronohealth.

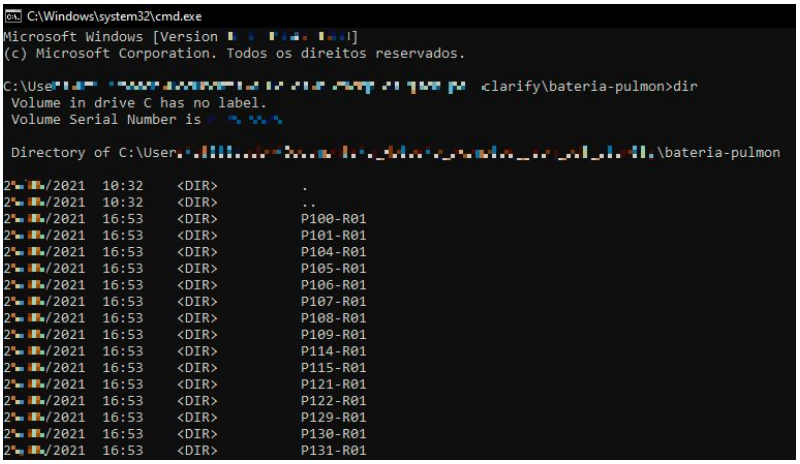
Those reports are exclusively generated by Kronohealth and only then are made available to the Clarify partners. In the scope of this project, the only responsibility of the reports are integrating them into the Web application; in order to perform this procedure properly, some key points should be carried out:

1. **Amount of data** At the date of writing this document, approximately 160 MB of files; spread across 100 folders with a total of approximately 400 files.

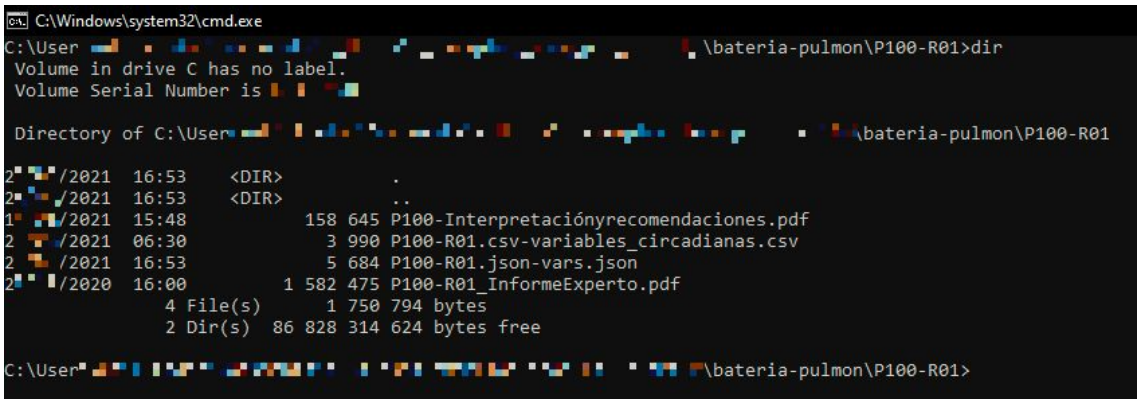
2. **Organization and typology of data**: The reports are organized in the main folder that defines the oncological hazard type with a series of folders organized under

[16] In order to maintain consistency, it was used the same address that was used for the project created on the Google Cloud Platform.
[17] Pentaho Transformation step that allows the user to modify columns as variables in plain JavaScript.
[18] http://www.kronohealth.com/

unique patient identifiers; each patient folder has four files, consisting of 2 PDF reports, 1 JSON, and 1 Comma-separated Values (CSV) file. The PDF reports are the final product where every parameter and variable can be found under analysis, patient info, and analysis of results accordingly. Figure 4.23 shows the described structure of folders.



(a) Main directory.



(b) Example of file structure on a given directory.

Figure 4.23: File structure of Kronohealth files.

3. **Utilization of the data in the Web application**: The Kronohealth reports will be available for the patients that joined the wearables utilisation under the individual analysis menu. Also, the data will go through Cloud Functions' processing to match the patient with the correct data.

4. **Data update and loading**: The Kronohealth partner will make the data available whenever there is a volume of data that justifies it, and it will be made in a secure way since the reports contain clinical data. Because the provided data is in a structured format of directories, it was chosen to load it into the project infrastructure to Cloud Storage buckets, making the data easily accessible inside the project ecosystem. Listing 4.9 presents the necessary instructions to upload the files to a bucket.

```
#Authentication in the designated project
$ gcloud auth login

# Copy the local files to a Cloud Storage bucket, where DIR is the local
    directory that contain the files and kronohealth is the unique name
    per project/zone
$ gsutil -m cp -r C:\DIR\*  gs://kronohealth

```

Listing 4.9: File upload to bucket.

When writing this document, the described instructions were being made manually whenever the partner informs that there is more data to be imported into the project's ecosystem.

4.2.3 Medical questionnaire

The medical questionnaire consists of a Google Forms[19] built with questions formulated by the medical staff. As it is known, Google Forms automatically integrates with Google Sheets[20] making the questionnaire responses available in a spreadsheet format.

Since that data is available in a tabular format, it is easier to access and analyse. The questionnaire is separated into a few sections with questions. Those can vary from the open response, numeric responses (scale), or multiple-choice questions. Currently, it is only possible to deal with numeric and multiple-choice questions.

The responses to the questionnaire are stored, by default, in Google Sheets. Thus, the storing is guaranteed by the Google Drive storage service. In order to maintain coherence, the proprietary of the questionnaire and the spreadsheet should be configured with the same email address that the project was created with. The information in the spreadsheet can be accessed, analysed, and modeled according to the needs simply by granting access to the spreadsheet to a project service account.

Once this is done, the spreadsheet can be accessed by the Google Cloud Functions and, in turn, the data curated.

When it comes to loading the data, this depends on doctors' intervention since those should share the questionnaire with the patients.

A few dozens of responses make the data relatively easy to access and process when writing this document. However, it is expected that those significantly increase, it is not anticipated any scalability or performance issues since Google Drive is a distributed system itself.

The medical questionnaire will be accessible in the Individual analysis menu, and there are two options available:

1. Consult previously answered questionnaires for a patient;

[19] https://docs.google.com/forms
[20] https://www.google.com/sheets

2. Make a new questionnaire or provide the link for answers.

4.2.4 Container optimised VM

In the Clarify project, the different partners provided different data sources and different methods to integrate them into the web application. As so, one of the data sources to integrate was a Docker container[21]; the provided docker contains a series of machine learning algorithms that will be integrated into the web application as an external service and provide a sequence of predictive models. Since the partners used the same database as an entry point to build them, the patient's unique identifiers will match with the available ones across the application. Thus, providing the clinicians with the capability to query across every possible service available on the web application in a coherent fashion.

In order to deploy a container in the project ecosystem, it was chosen to deploy yet another Virtual Machine, this time with the Container optimised Operating System that is an operating system image for Compute Engine VMs that is optimised for running Docker containers [28].

In order to do so, it is necessary to use the Container Registry on the Google Cloud Platform to import the Docker container into the project. If the importation is successful, the container image should be ready for use as presented in Figure 4.24.

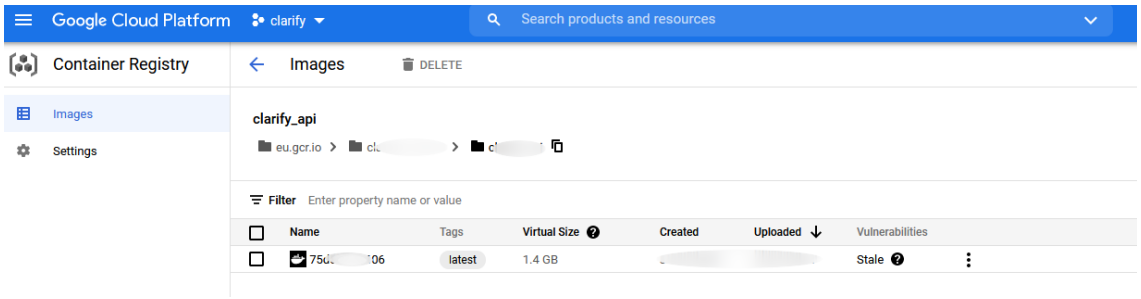


Figure 4.24: Container Registry menu with successfully imported Docker image.

Since the container image is in the project ecosystem, the conditions to deploy the VM are in place, and a similar procedure to the one discussed in section 4.1.4 should be followed, with the particularity that the previously imported container image should be selected, and the Boot disk changed to the Container optimised Operating System (OS).

In an attempt to verify the proper functionality of the VM, the instructions in Figure 4.25 should be done and returned the corresponding responses.

In Figure 4.25 is guaranteed that the previously imported image "clarify_api" is up and running. As it is possible to observe in Figure 4.25b the curl request is made to the IP "10.132.0.4:5000" which is the internal GCP resource IP and is only accessible on the VPC, i.e., only accepts requests from resources inside the project ecosystem. Also, it is possible to notice that both the request and the response are in JSON format, which

[21] https://www.docker.com/

means that the Computer optimised VM is set up to function as an API, only accepting incoming requests from within the project ecosystem.

```
slcg-api2 /home/henrique_joaquim # docker ps
CONTAINER ID   IMAGE                                                  COMMAND                  CREATED          STATUS          PORTS     NAMES
629d12f5eb68   gcr.io/stackdriver-agents/stackdriver-logging-agent:1.8.9   "/entrypoint.sh /usr…"   11 minutes ago   Up 11 minutes             stackdriver-logging-agent
fb9eaa83cc38   eu.gcr.io/clarify-[redacted]/clarify_api                "/bin/sh -c 'gunicor…"   11 minutes ago   Up 11 minutes             klt-slcg-api2-ljfc
```

(a) Running Docker instance.

```
slcg-api2 /home/henrique_joaquim # curl -X POST "http://10.132.0.4:5000/clarify/[redacted]/v1/tabular/predict" -H "accept: application/json" -H "Content-Type: application/json" -d  @test_patient_features.json
{"data":[{"label":"1","predict_period":6,"probability":0.857952619280453},{"label":"1","predict_period":12,"probability":0.9589197131047339},{"label":"1","predict_period":18,"probability":0.7563683815775629},{"label":"1","predict_period":24,"probability":0.9494334478690702},{"label":"1","predict_period":30,"probability":0.7957786783224152},{"label":"1","predict_period":36,"probability":0.8369387738397371},{"label":"1","predict_period":42,"probability":0.846082257577236},{"label":"1","predict_period":48,"probability":0.8720525937192173},{"label":"1","predict_period":54,"probability":0.9682995439938512},{"label":"1","predict_period":60,"probability":0.9555478414281922},{"label":"1","predict_period":66,"probability":0.8273441163240938},{"label":"1","predict_period":72,"probability":0.8372300575474475},{"label":"1","predict_period":78,"probability":0.8218793032102799},{"label":"1","predict_period":84,"probability":0.7306269237955643},{"label":"1","predict_period":90,"probability":0.802596403207305},{"label":"1","predict_period":96,"probability":0.9268636429643159}]}
```

(b) Example request.

Figure 4.25: Verifying container optimised OS VM functionality.

4.3 Services - Google Cloud Functions

4.3.1 Google Cloud Functions

This section intends to describe the implementation of the services that were made available in the web application. The system architecture was designed with a microservices-like approach. Thus, those were implemented with the Google Cloud Functions, the GCP Function-as-a-Service (FaaS) available solution. As previously discussed, some of the available services were developed in collaboration with the Clarify partners. Therefore, in this project's scope, it will only describe the core functionalities of each deployed Cloud Function. The main difference between the Cloud Functions and the previously developed partners' models is the access to the data. Previously, the partners had their own sources however, migrating to the integrated project data sources was necessary. In order to do that, an extensive iterative process was done to guarantee the model's integrity.

Of note, every communication with the Cloud Functions is done using JSON, i.e., the requests are expected to be in that same format and the provided response. Before any computation, a user authentication validation and a request validation are performed to check if every needed parameter is received. Regarding the access to the data structures, in order to enable secure communication between the Cloud Functions and the previously described structures, it was necessary to configure a Secret Manager[22] [34].

Below, a list of the developed Cloud Functions:

1. **cox_survival_curve**: compute the Cox Model; this GCF is fed with data directly from the database; it is invoked from the web application due to user interaction. The Cloud Function receives the parameters in analysis such as hospital, timelapse, stages, and gender. As a response, it sends back to the UI a JSON with the coordinates of the curves[23].

[22] Secret Manager allows to adequately store sensitive information such as passwords, keys, and certificates.

[23] The JSON is then handled and presented by React.

2. **kh_survival_curve**: compute the Kaplan-Meier Estimator; this GCF is fed with data directly from the database; it is invoked from the web application and due to user interaction. The Cloud Function receives the parameters in analysis, such as gender and stage. As a response, it sends back to the UI a JSON with the coordinates of the curves.

3. **dataset_info**: query the database to obtain statistical information about the required dataset, i.e., providing the statistical data from the in analysis population.

4. **descriptive_analysis**: query the database to obtain a descriptive analysis according to the required selected dataset and required timelapse.

5. **general_api_handler**: this Cloud Function was developed from scratch to integrate external services with the developed web application; it receives requests from the UI and redirects them to the intended API, redirecting the response back to the UI.

6. **kronohealth**: the goal is to provide the data described in section 4.2.2, a unique patient identifier is provided to the function, and it responds with the reports available for that patient.

7. **patient_info**: the Cloud Function is responsible for providing demographic information about a single patient upon request.

8. **radar**: computes a series of scalar variables from the data described in section 4.2.3 in order to build a radar graph[24].

9. **spreadsheet_reader**: queries the medical questionnaire with the required patient identifier and returns the responses to the questionnaire.

4.3.2 Google Cloud Functions optimisation

During the implementation of the Google Cloud Functions, it was noticed that the response time for those increasingly longer depended on the amount of data. Thus, the necessity to do performance optimisations arose. The GCF are stateless and operate in execution environments. Those are often initialized from scratch, which is known as "Cold Start". The cold start can reveal itself problematic since it is unavoidable, and the request to a GCF in the cold start will inevitably take longer. The cold state[25] can be due to one of the following:

1. **Not triggered function**: the function was successfully deployed, but it was not called yet;

[24] As mentioned, the response from the GCF is always a JSON, in this case, is no different, the UI then processes the response JSON and finally renders as a radar graph.
[25] State of a Cloud Function that will be initialized in cold start.

2. **Recycled function**: the function has not been invoked for a certain period[26].

3. **Scaling**: the execution environment reached its request limit, and it was necessary to create a new instance.

According to [26], the solutions for this issues go through:

1. **Dependencies**: packages need to be loaded to the execution environments; the more significant the volume and quantity of dependencies, the longer the cold start time. With that being said, it should be considered the following:

   a) Analysis of which dependencies are strictly necessary;

   b) Dependency version: because all GCF dependents share the dependency cache, users and deployments can reuse the most popular versions of a module [51];

2. **Global variables**: GCF often recycles the execution environment of a previous invocation; if a variable is declared in global scope, its value can be reused in subsequent invocations without computation.

3. **Lazy loading**: considering the previously mentioned solutions, lazy loading can be done on both; not every dependency and global variable are used in all code paths. Thus, those resources can be lazily initialized on demand.

4. **Cronjobs**: a slightly bolder solution is proposed in [52]; using cronjobs to invoke the GCF on regular time intervals in order to prevent the cold start. Although this could potentially solve the issue, it adds additional resources to the project, translating into extra costs.

Regarding the actual implementation of optimisations to the GCF, it was decided that in the scope of the projects and according to the developed ones to act on the dependencies and global variables issue. Concerning dependencies, a careful analysis was done on which ones should be used for each Cloud Function; also, a web crawler was developed recurring to Scrapy[27]. The goal with the web crawler is to recursively check the used dependencies in each Cloud Function and crawl the PyPi[28] to verify if it is as updated as possible. If that is not the case, the developed software is set to send an email to a configured address. When writing this document, this process is scheduled to run once a week as a shell script through the cron command-line utility on the implemented VM.

Concerning the global variable initialization, the most resource-consuming operation in the developed Cloud Functions is the instancing of the database and, in turn, query it. The Cloud Functions were modified to keep the database instance as global and the query' resulting data frame, as illustrated by the example code in Listing 4.10.

[26] Typically this period varies from fifteen to twenty minutes.
[27] https://scrapy.org/
[28] https://pypi.org/ - Python Package Index

```
# Global scope
db = clarify_db()
with db.connect() as conn:
    df = pd.read_sql(  select   * from table where condition , conn)

# Entry point scope
#    - similarly to try catch syntax
#    - database lazy loading
def entry_point_function(request):

    global db
    if not db:
        db = clarify_db() # database instancing specific computation
    ...
```

Listing 4.10: Example of database lazy loading.

The optimisation results will be further discussed in section 5.1.

4.4 User Interface

As previously discussed, the React framework was used to develop the User interface. React is a JavaScript library created to build scalable and fast Web applications. It consists of a Single Page Application, a development feature that allows rendering only the desired content on a web page instead of reloading the entire application to present new information. This feature is vital since the project is aimed at a Big data context. Presenting enormous amounts of data on a Web page might slow down the application, and the React method of operation can provide the tools to address this issue. Consequently, it does not incur the risk of having the page freeze while loading data, leading to a smoother user-system interaction and better performance, highly improving the UX.

It followed the Atomic Design principles, separating the application modules into smaller components to reduce complexity and increase abstraction, which improves the development process. When using React, the frontend is only responsible for rendering the components while all the logic and rules are addressed in the backend, concretely, by the Cloud Functions. The presentation of data is gradual and done as the end-user makes requests. Figure 4.26 presents the component organization, which can be thought of as menus. It starts from the main menu (MainCard) and gradually progresses to more specific menus as the level of React components decreases.

Initially, in order to access the Homepage, the user is prompted with the required authentication. Once the authentication is surpassed, the user is presented with three distinct menus: "Individual Analysis", "Population Analysis,"and "Knowledge Graph". Each of those three menus has its specific submenus that, in turn, have several options. As demonstrated in Figure 4.26, the “get“ represents the functionality responsible for the UI - Cloud Functions interaction.

Detailing the User Interface:

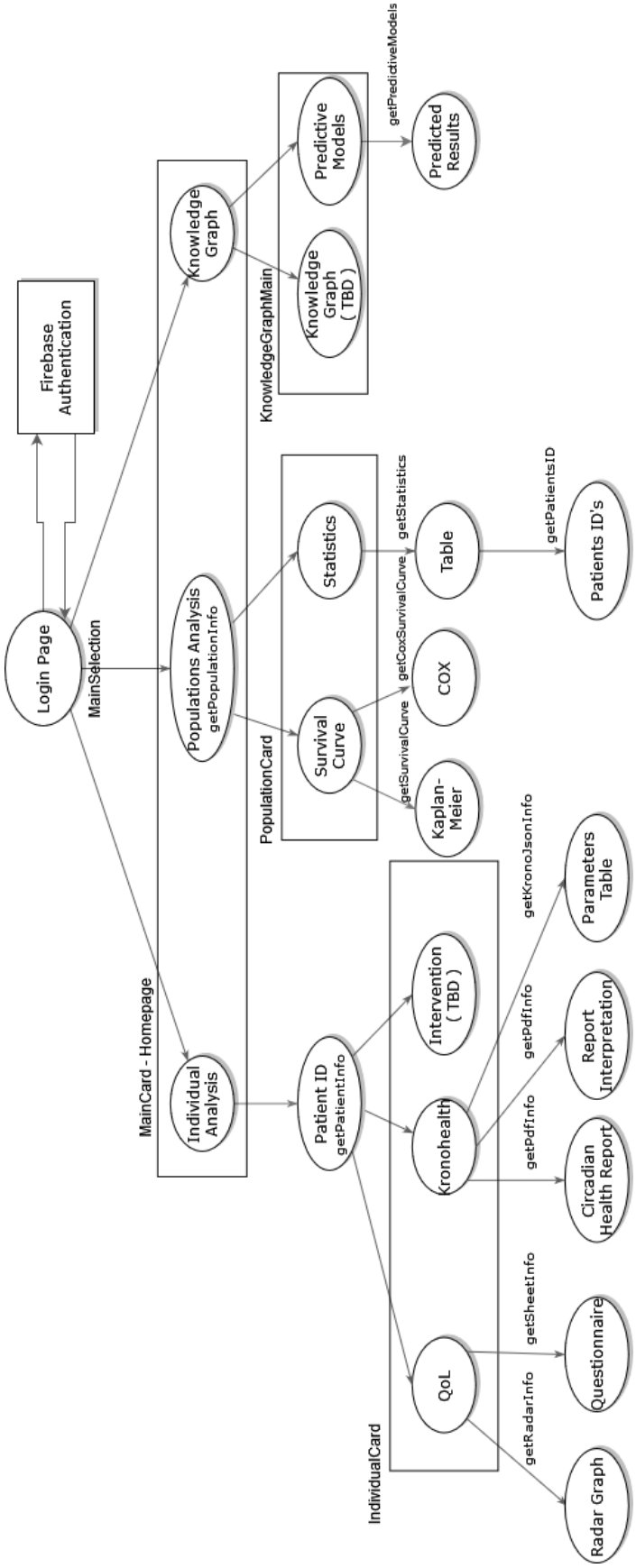


Figure 4.26: UI organization.

1. **Login Page**: it is the entry point for the application. It is provided to the user a simple interface to log in with email and password. A form with the pair is submitted, and Firebase handles the authentication via API.

2. **Home Page**: after the login is successful the user is redirected to the home page. On this page, the user can choose the type of oncological hazard and hospital. It presents three distinct options to proceed: Individual Analysis, Population Analysis, and Knowledge Graph.

    a) Individual Analysis: The user is requested to introduce a unique patient identifier to complete the analysis on this component. Afterward, it is presented general demographic information on that patient and three options: Medical Questionnaire, Kronohealth Analysis, and Intervention

        i. Medical Questionnaire (QoL): it presents the options regarding the medical questionnaire: Radar Graph Analysis and Questionnaire Responses. Also, the user can start a new questionnaire.

            A. Radar Graph Analysis: presents the classification that the patient obtained through the questionnaire against the average, maximum and minimum classifications of patients.

            B. Questionnaire Responses: presents the patient responses to previously done questionnaires.

        ii. Kronohealth Analysis: provide the Kronohealth files prevenient from the integration with the Clarify partner wearable devices. On this component, the further options available are Circadian Health Report, Report Interpretation, and Parameters Table.

            A. Circadian Health Report: complete patient data analysis report, the report is presented as a PDF preview, and it is also given the option to download it.

            B. Report Interpretation: report for results interpretation, the report is presented as a PDF preview, and it is also given the option to download it.

    b) Population Analysis: the user is presented with information about the dataset, such as the total of patients, population percentage in each stage, minimum, maximum, and average population age. Also, there are two options available: Survival Curve and Descriptive Analysis.

        i. Survival Curve: the user can choose between the available estimators, Cox Model and Kaplan-Meier. In order to plot the curves, the user is requested to select the intended diagnostic stages and population gender; both are not mandatory to make the request.

c) Descriptive analysis: this feature shows the descriptive analysis of the selected dataset regarding the number of patients, mean age, median age, and survival time in days, months, or years. For authorised users, it is also possible to download a CSV file with the patients' identifiers for the desired presented table.

3. Knowledge Graph: the user can access external services developed by the Clarify partners. The options available are Predictive Models and Knowledge Graph.

a) Predictive Models: the user must submit a patient identifier to get a response from the service.

b) Knowledge Graph: at the time of writing this document, this feature is still under development; the goal is to use the component to integrate other Clarify partner services.

5

Results

This section presents the optimisation results for the Cloud Functions and the overview of the global results for the implemented application.

5.1 Google Cloud Functions Optimisation Results

As discussed in section 4.3.2, in order to provide a better UX the developed Cloud Functions were lacking the due optimisations. In order to confirm if the developed optimisations had an effect in the performance of the Cloud Functions, a series of testing was done. The Functions that were drastically affecting the UX have been chosen to perform the test, concretely: survival_curve[1], dataset_info and patient_info. Although the optimisations were done for every Cloud Function.

The adopted methodology for testing was as follows:

1. **Conditions**:
    a) Anonymous navigation (in order to avoid cache effects);
    b) User logged in.
2. **Procedure**:
    a) Force cold-start: re-deploy GCF (if testing the cold start response);
    b) Call in analysis GCF on the Web application;
    c) Retrieve the execution time from Google Cloud Platform.
3. **Keynote**:
    a) Google does not provide any information about the uncertainty;
    b) The testing was performed by a single user, maintaining the same conditions across testing;

[1] The results are analogous for Cox and Kaplan-Meier estimators.

c) GCF works on demand, so it is expected that with the increase in request volume, performance improves.

Performed tests:

1. Cold Start response:
   a) Before optimising;
   b) After optimising.
2. Warm response:
   a) Before optimising;
   b) After optimising.

Detailed results for each test can be found in Annex II. The obtained results can be visualised in Figures 5.1, 5.2 and 5.3.

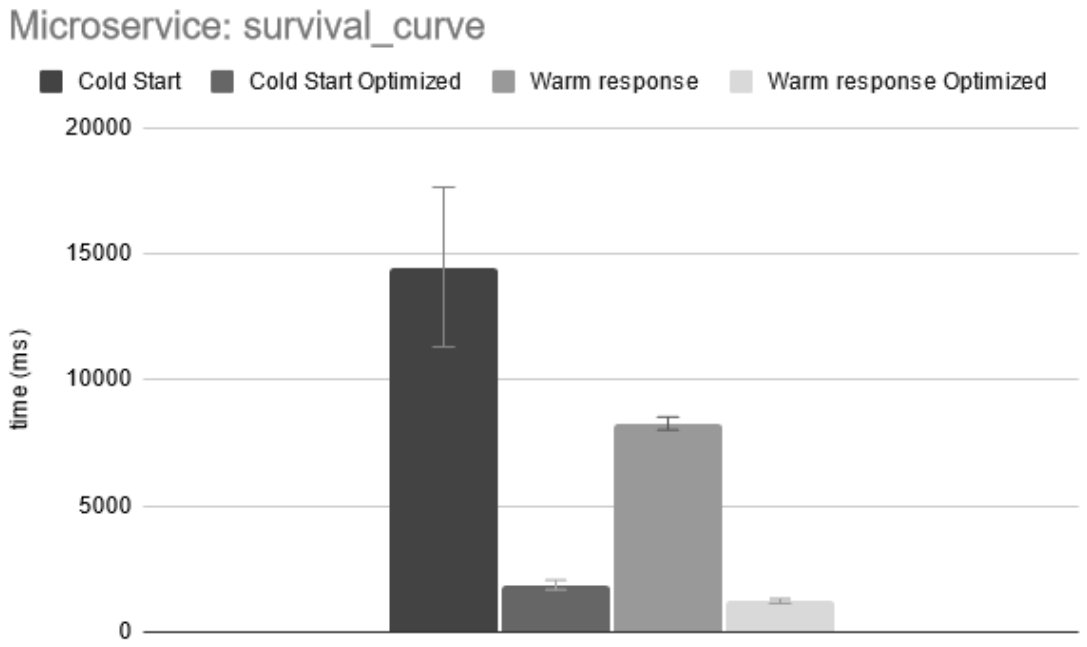


Figure 5.1: Results on cold start and warm response before and after the optimisation for the survival_curve Cloud Function.

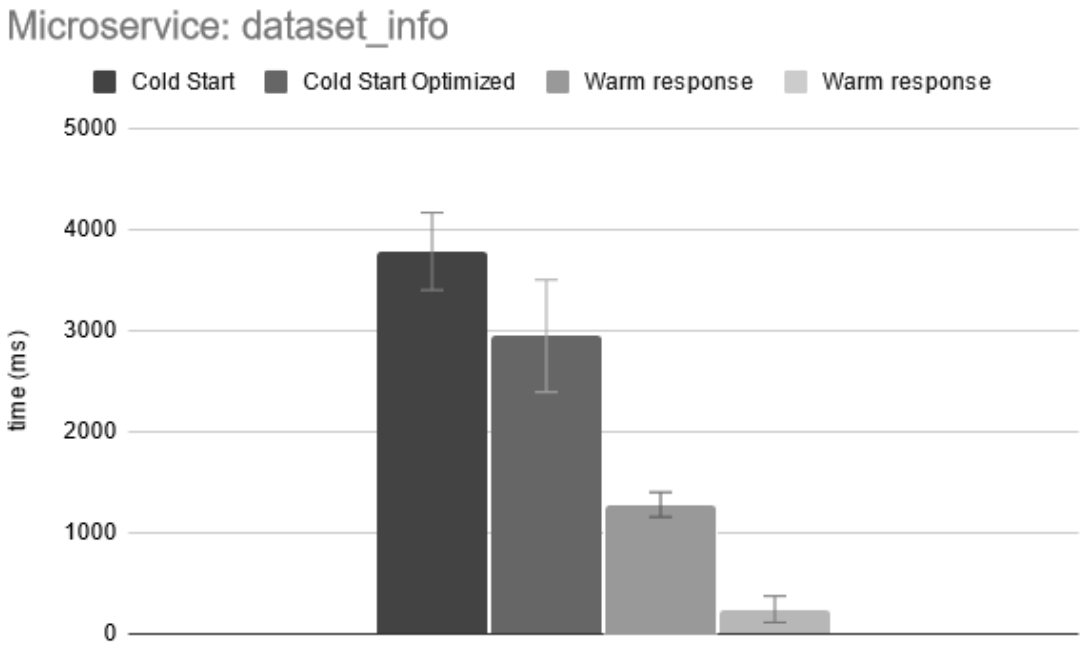


Figure 5.2: Results on cold start and warm response before and after the optimisation for the dataset_info Cloud Function.

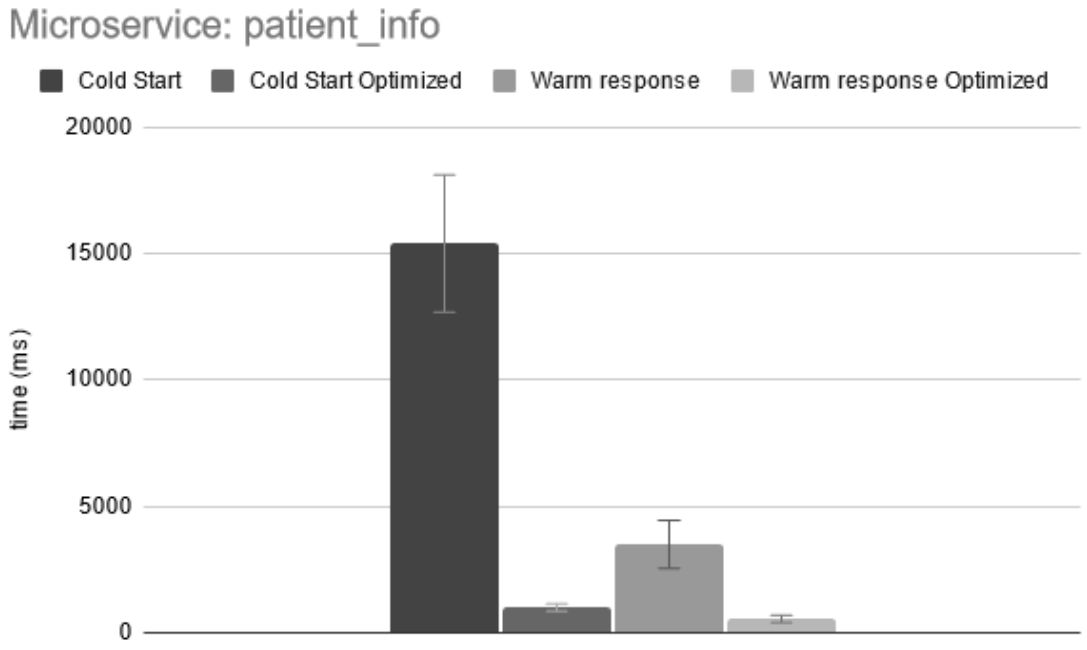


Figure 5.3: Results on cold start and warm response before and after the optimisation for the patient_info Cloud Function.

As it is possible to see, the optimisations made by the Cloud Functions have a dramatically lower response time. This result turns out to be very important because it dramatically improves the user experience and, consequently, the platform's usability.

As seen, before the optimisations, some of the Cloud Functions had response times between 15 and 20 seconds, which for a web application is unthinkable. With the reduction in response times, we are now facing very good and perfectly acceptable conditions considering the project's paradigm.

It is also important to mention that from the tests carried out with the end-users, they were very impressed and satisfied with the response times of the available services.

5.2 Web Application Results Overview

As an analysis of the overall results of the implementation of the architecture and consequently of the web application developed, the different screens are presented.

Figure 5.4: Login page.

The Login page was successfully developed, recurring to React and Firebase authentication. In order to perform a login, the user must be registered by an authorised entity directly in the Firebase console, thus ensuring the security and confidentiality of the data displayed inside the application.

Using the Firebase authentication, it is possible to avoid further development of a secure authentication method. Since we are using this external provider that is used by multiple organizations of the most diverse sizes and therefore has diverse applications with the most distinct data of different complexities and that require different types of care and security.

It is ensured that the developed application has all the requirements regarding authentication, benefiting, once again, from the fact that the architecture is built on the

Google Cloud Platform. It also avoids additional development that, when not developed by security experts, can eventually leave security flaws and compromise the data, which in this case, as they are clinical data, are pretty sensitive and may even compromise the entire application.

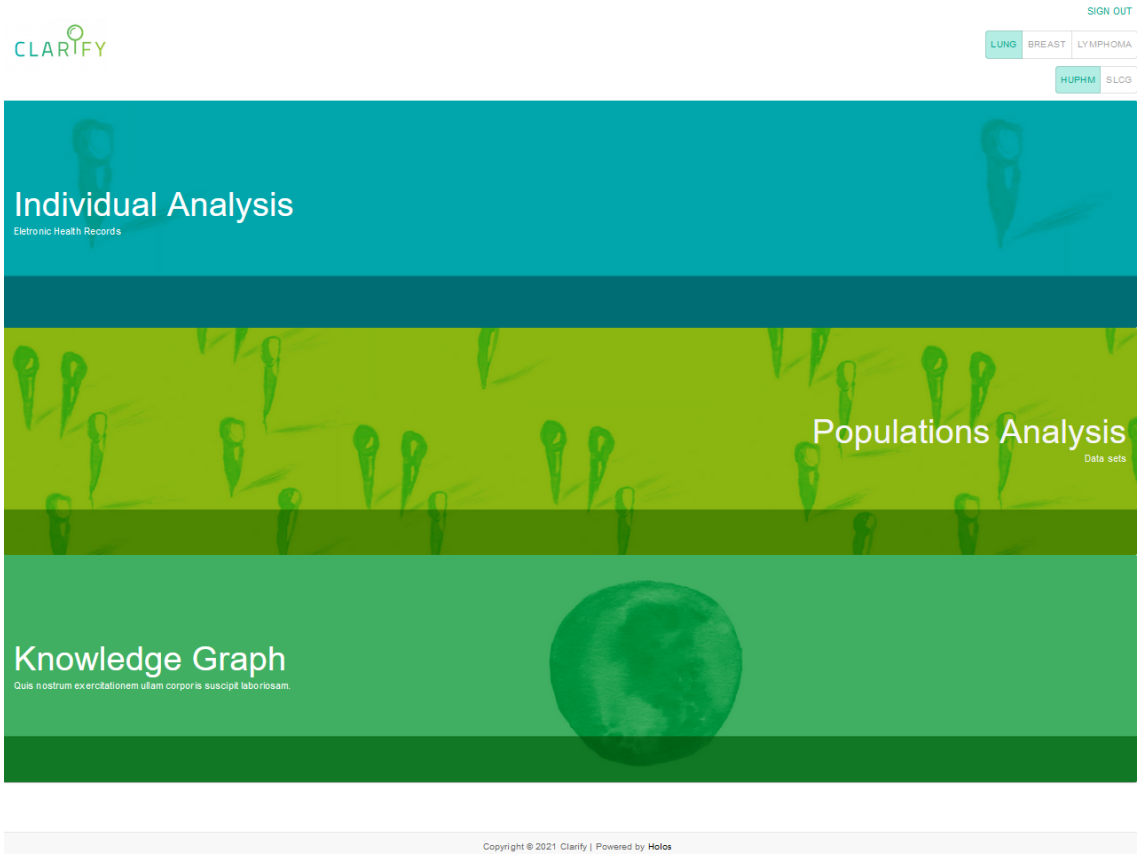


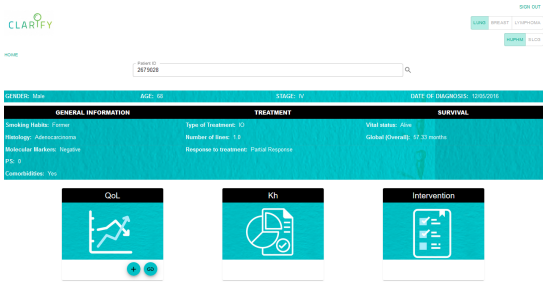
Figure 5.5: Homepage.

On the homepage, the user has three options, also the possibility to choose between the different types of oncological hazard and the available hospital, upon configuration[2].

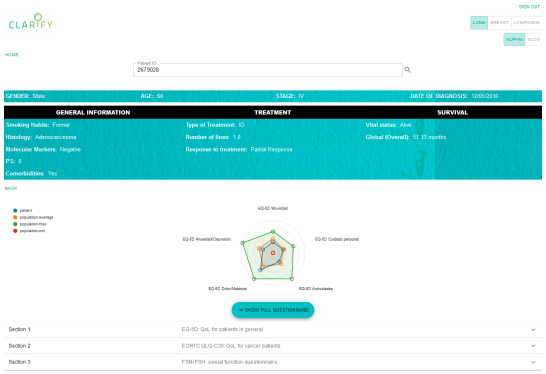
An entity carries out the configuration of both types of cancer and hospitals with permissions. This process is done manually. Therefore, it is also guaranteed that in a future scenario, when there are the most diverse users, in this case, authorized. However, different hospitals only have access to the data that is their responsibility, ending up safeguarding non-legitimate access to data for reasons of competitiveness.

From the application's perspective, the choice of hospital and oncological hazard definitions is saved in dynamic variables that are handled by React. These will later be used for requests made by the user. With this approach, the user experience is greatly improved as the user does not have to do any other action besides placing the request. On the other hand, this also becomes more efficient when requesting fulfillment since the variables are instantly ready for submission.

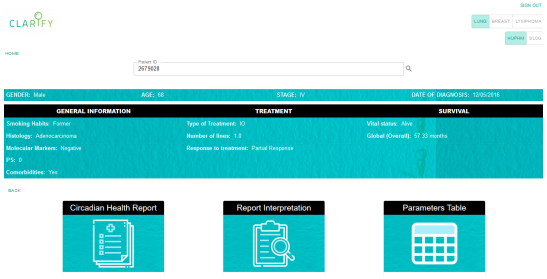
[2]Permissions vary from user to user



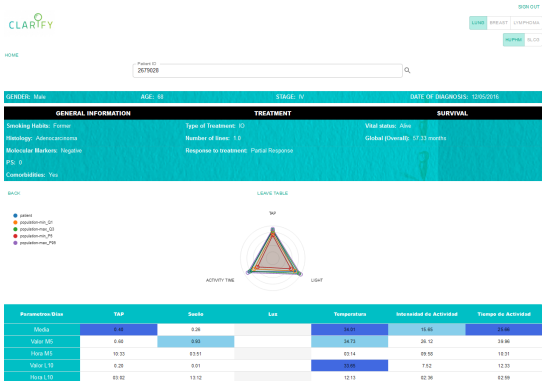
(a) Individual analysis main page.



(b) Medical Questionnaire page.



(c) Kronohealth analysis page.



(d) Kronohealth - parameters table page.

Figure 5.6: Individual Analysis

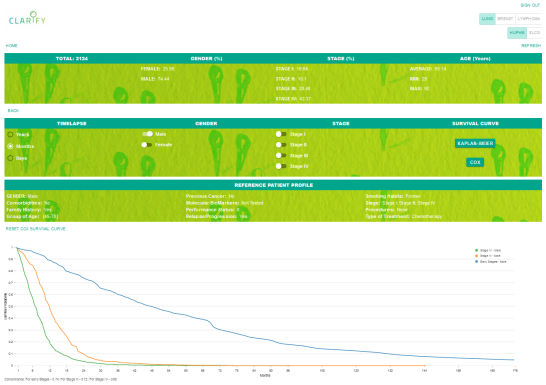
However, all requests to Cloud Functions are protected, whether in React development or in the Cloud Functions themselves, by checks that only allow the computation of the request if all the conditions for this are met. Thus, obtaining the guarantee that the results obtained are scientifically correct is of extreme importance since the application operates in a real-life healthcare context. From a computing perspective, it is also guaranteed that the Cloud Functions developed are idempotent.

Regarding the Individual analysis, it was possible to join several different sources and integrate them in a single page with an easy-to-understand interface. With a few clicks, it is possible to go through all the available information for a single patient while maintaining simplicity and at the same time presenting complex information.

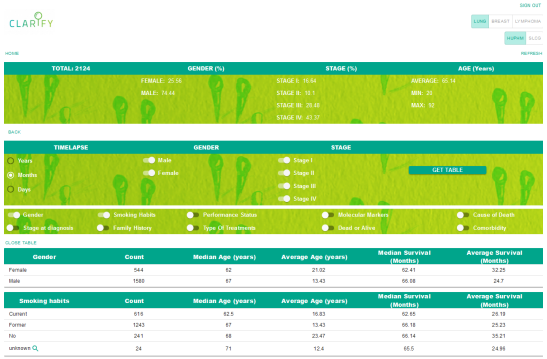
The individual patient analysis is also designed to be extremely fast and straightforward to use, whether clinicians intend to use it during an appointment and therefore with the patient in front of them or for further analysis. In tests carried out with the clinicians themselves, they expressed great satisfaction about the developed interface and how information is presented and organized.

Another essential point to mention is that the information presented by the web application in this context is mischaracterized as much as possible. Therefore, unless the physician has the list of unique identifiers to connect the patient and the identifier, it

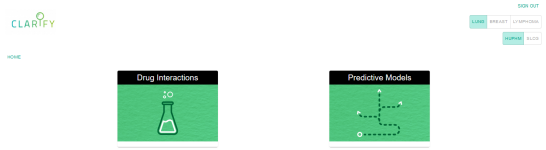
(a) Population analysis main page.



(b) Descriptive analysis page.



(c) Survival curves - Cox Estimator.



(d) Survival curves - Kaplan Meier Estimator.

Figure 5.7: Individual Analysis

becomes complicated to conclude someone's identity. In this way, it is guaranteed that sensitive information is not exposed which from the point of view of Clarify partners also ends up being essential legal protection.

Regarding population analysis, it was possible to integrate the intended data sources, in this case using a relational database, which in turn feeds the Cloud Functions for the construction of models and the descriptive analysis of the population.

The developed interface is straightforward to use. Any available models can be obtained with a few clicks, thus empowering clinicians, healthcare professionals, and other project partners, such as data scientists, analysts, and others, to quickly and effectively compute complex models.

Population analysis turns out to be of great importance either to understand the population's behavior as a whole or even to analyse isolated cases, with the possibility of knowing, easily and quickly, how the population behaves, doctors also end up being better able to make better decisions.

6

Conclusion and Future Work

The development of the project described throughout this document was only possible due to the mutual help of Clarify partners. An architecture from scratch and a web application was developed in a serverless paradigm.

To date, it is operating in a real-life context and has given the necessary answers to the requirements imposed by its users.

It is believed that the fact that the application was developed in a serverless context and was supported by a microservices-like architecture made the development process more manageable and, at the same time, overcome complicated constraints inherent of traditional architectures, from server maintenance to the development of secure authentication methods and above all about the scalability of the resources used. Being the project at an initial stage, the serverless approach guarantees that the application developed has all the conditions to scale up progressively without constraints, whether this growth is the number of users, whether in the amount of data, or in the number of new data sources to be integrated.

An ergonomic, user-friendly, and responsive tool was developed, capable of being used comfortably on any device and without prior knowledge on how the application works. With this project, clinicians, the end-users of this tool, were provided a tool to obtain, analyze and communicate scientific information about their patients. These same clinicians initially generate these data, giving rise to an increasingly positive feedback loop in which the better and more voluminous the data that reaches the application, the better the final result will be. That is, the more valuable the application will also be.

It is also important to mention that it is a colossal gratification to be involved in a project of such magnitude in which doctors, data scientists, and engineers collaborate with the common goal of solving a problem that affects society. Verifying that there are indeed tangible results in the way the clinicians' approach is made and that, ultimately, it manages to have a tremendous and positive impact on people's lives through science and technology.

6.1 Future Work

During the development of this project, there were found several opportunities for improvement, as is typical in every software application.

As mentioned, the application at the time of writing this document has not yet attained a Big data paradigm, and for that reason, it was used a relational database to store the hospital data. However, with the increase of data, such an approach will eventually turn out inefficient in the future. This situation can be overcome by the use of the GCP service Google Big Query, which also enables the use of SQL-like queries in order to retrieve data while adding the capabilities of Apache Spark or Hadoop [32], and for that reason would eventually be the adequate solution and the one that requires fewer changes in the current system. At the time of writing this document, the Kronohealth files are being loaded into the application manually upon request. However, a hypothetical solution for the automation of such processes would be the use of the Vendor dropbox, which enables the partner to submit the files directly to the project without to option to access any other resources [35].

Another possible improvement regarding the data loading process would be to stop using Pentaho ETL, which revealed an extremely efficient and easy-to-use tool, in exchange for Google Cloud Dataflow. Cloud Dataflow is a fully controlled ETL service based on Apache Beam that is supported by Google and can be used to run a serverless ETL pipeline built on Google ecosystem components [29]. The use of such a tool would eventually facilitate the orchestration of programmed events, centralization of logs, and decrease the use of software outside the GCP ecosystem.

Another topic that leaves room for further research is the use of Cloud Run over App Engine. Cloud Run runs containers, so it is necessary to build a container for each release and push it to GCP. Unlike App Engine, Cloud Run only runs when requests come in, so it has no cost for time spent idling. However, containerized apps are more portable but not something to focus on during development [14].

Another interesting topic that encourages further investigation is the performance of serverless resources. According to [8], and [9], the performance of some GCP resources can significantly be affected by the location in which resources are deployed and the preferred programming language used for development. As mentioned above, the used programming language for the development of GCF was Python. However, it has an unsatisfactory performance. On the other hand, Go[1] which is a programming language developed in the cloud for the cloud and seems to be very well adapted to the serverless environment and microservices architecture [9] which could potentially be a perfect match with the project in hand.

Finally, the creation of a differentiated test environment and production environment in order to keep production data integrity or avoid affecting the use of the application during the development process. Still regarding the development process, the creation

[1] https://go.dev/

of unit testing in order to assure quality and detect bugs before new developments reach the production environment.

Bibliography

[1] *Apache spark™ - unified analytics engine for big data*. URL: https://spark.apache.org/ (cit. on p. 7).

[2] *AWS Lambda Documentation*. https://docs.aws.amazon.com/lambda/index.html. (Accessed on 09/11/2021) (cit. on p. 12).

[3] *AWS vs. Azure vs. Google Cloud | Top Cloud Providers 2021*. https://www.datamation.com/cloud/aws-vs-azure-vs-google-cloud/. (Accessed on 09/10/2021). Aug. 2021 (cit. on p. 9).

[4] *AWS Vs. Azure Vs. Google Cloud Platform | by Fintelics | Medium*. https://fintelics.medium.com/cloud-platform-comparison-aws-vs-azure-vs-google-cloud-c956a9033282. (Accessed on 09/10/2021). Apr. 2021 (cit. on p. 9).

[5] *Azure Functions documentation | Microsoft Docs*. https://docs.microsoft.com/en-us/azure/azure-functions/. (Accessed on 09/11/2021) (cit. on p. 12).

[6] K. Bilal et al. "Trends and Challenges in Cloud Datacenters". In: *IEEE Cloud Computing* 1.1 (2014), pp. 10–20. ISSN: 23256095. DOI: 10.1109/MCC.2014.26 (cit. on p. 8).

[7] B. Blamey. "A Comparative Study of Big Data Frameworks Related papers". In: () (cit. on pp. 6, 7).

[8] G. Blaquiere. *Cloud Run and Cloud Functions: Does the region change the performances?* https://medium.com/google-cloud/cloud-run-and-cloud-functions-does-the-region-change-the-performances-b967e5cee0cc. Oct. 2021 (cit. on p. 69).

[9] G. Blaquiere. *Serverless performance comparison: Does the language matter?* https://medium.com/google-cloud/serverless-performance-comparison-does-the-language-matter-c72a7191c799. Nov. 2021 (cit. on p. 69).

[10] F. Cassidy. *A day in Data*. https://www.raconteur.net/infographics/a-day-in-data/. July 2020 (cit. on p. 1).

[11] P. Chandarana and M. Vijayalakshmi. "Big data analytics frameworks". In: *2014 International Conference on Circuits, Systems, Communication and Information Technology Applications, CSCITA 2014* (2014), pp. 430–434. DOI: 10.1109/CSCITA.2014.6839299 (cit. on pp. 1, 6).

[12] M. Chen, S. Mao, and Y. Liu. "Big data: A survey". In: *Mobile Networks and Applications* 19.2 (2014), pp. 171–209. ISSN: 1383469X. DOI: 10.1007/s11036-013-0489-0 (cit. on pp. 6, 8).

[13] *Cloud Functions documentation | Cloud Functions Documentation*. https://cloud.google.com/functions/docs. (Accessed on 09/11/2021) (cit. on p. 12).

[14] P. Craig. *Cloud Run vs App Engine: a head-to-head comparison using facts and science*. https://dev.to/pcraig3/cloud-run-vs-app-engine-a-head-to-head-comparison-using-facts-and-science-1225. Nov. 2020 (cit. on p. 69).

[15] S. Dash et al. "Big data in healthcare: management, analysis and future prospects". In: *Journal of Big Data* 6.1 (2019). ISSN: 21961115. DOI: 10.1186/s40537-019-0217-0. URL: https://doi.org/10.1186/s40537-019-0217-0 (cit. on pp. 5, 8).

[16] T. Ercan. "Effective use of cloud computing in educational institutions". In: *Procedia - Social and Behavioral Sciences* 2.2 (2010), pp. 938–942. ISSN: 18770428. DOI: 10.1016/j.sbspro.2010.03.130. URL: http://dx.doi.org/10.1016/j.sbspro.2010.03.130 (cit. on p. 8).

[17] M. Favaretto et al. "What is your definition of Big Data? Researchers' understanding of the phenomenon of the decade". In: *PLoS ONE* 15.2 (2020), pp. 1–20. ISSN: 19326203. DOI: 10.1371/journal.pone.0228987. URL: http://dx.doi.org/10.1371/journal.pone.0228987 (cit. on p. 2).

[18] K. Feldman, D. Davis, and N. V. Chawla. "Scaling and contextualizing personalized healthcare: A case study of disease prediction algorithm integration". In: *Journal of Biomedical Informatics* 57 (2015), pp. 377–385. ISSN: 15320464. DOI: 10.1016/j.jbi.2015.07.017. URL: http://dx.doi.org/10.1016/j.jbi.2015.07.017 (cit. on pp. 1, 3).

[19] M. F. Furukawa et al. "Despite substantial progress in EHR adoption, health information exchange and patient engagement remain low in office settings". In: *Health Affairs* 33.9 (2014), pp. 1672–1679. ISSN: 15445208. DOI: 10.1377/hlthaff.2014.0445 (cit. on p. 5).

[20] R. F. Gillum. "From papyrus to the electronic tablet: A brief history of the clinical medical record with lessons for the digital age". In: *American Journal of Medicine* 126.10 (2013), pp. 853–857. ISSN: 00029343. DOI: 10.1016/j.amjmed.2013.03.024. URL: http://dx.doi.org/10.1016/j.amjmed.2013.03.024 (cit. on p. 5).

[21] M. Golfarelli. "Open source bi platforms: A functional and architectural comparison". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 5691 LNCS (2009), pp. 287–297. ISSN: 03029743. DOI: 10.1007/978-3-642-03730-6_23 (cit. on p. 16).

[22] Google. *App Engine documentation | App Engine Documentation | Google Cloud*. https://cloud.google.com/appengine/docs. Nov. 2021 (cit. on p. 20).

[23] Google. *app.yaml Configuration File*. https://cloud.google.com/appengine/docs/standard/nodejs/config/appref. Nov. 2021 (cit. on p. 33).

[24] Google. *Authenticating Users*. https://firebase.google.com/firebase-and-gcp. Nov. 2021 (cit. on p. 43).

[25] Google. *Authenticating Users*. https://cloud.google.com/appengine/docs/standard/go/authenticating-users. Nov. 2021 (cit. on p. 43).

[26] Google. *Cloud Functions Documentation | Google Cloud*. https://cloud.google.com/functions/docs/bestpractices/tips#performance. Nov. 2021 (cit. on p. 56).

[27] Google. *Cloud SQL features | Cloud SQL Documentation | Google Cloud*. https://cloud.google.com/sql/docs/features. Nov. 2021 (cit. on p. 38).

[28] Google. *Container-Optimized OS from Google documentation | Google Cloud*. https://cloud.google.com/container-optimized-os/docs. Nov. 2021 (cit. on p. 53).

[29] Google. *Dataflow | Google Cloud*. https://cloud.google.com/dataflow. Nov. 2021 (cit. on p. 69).

[30] Google. *General-purpose machine family nbsp;|nbsp; Compute Engine Documentation*. https://cloud.google.com/compute/docs/general-purpose-machines. Nov. 2021 (cit. on p. 34).

[31] Google. *Google Cloud Overview*. https://cloud.google.com/docs/overview. Nov. 2021 (cit. on p. 30).

[32] Google. *Querying Cloud Bigtable data | BigQuery | Google Cloud*. https://cloud.google.com/bigquery/external-data-bigtable. Nov. 2021 (cit. on p. 69).

[33] Google. *Quickstart for node.js in the standard environment*. https://cloud.google.com/appengine/docs/standard/nodejs/quickstart#cloud-shell. Nov. 2021 (cit. on p. 33).

[34] Google. *Secret Manager conceptual overview | Secret Manager Documentation*. https://cloud.google.com/secret-manager/docs/overview. Nov. 2021 (cit. on p. 54).

[35] Google. *Sharing and collaboration | Cloud Storage | Google Cloud*. https://cloud.google.com/storage/docs/collaboration. Nov. 2021 (cit. on p. 69).

[36] Google. *What is Cloud Storage? | Google Cloud*. https://cloud.google.com/storage/docs/introduction. Nov. 2021 (cit. on p. 38).

[37] "Hadoop Consulting | HDFS Consulting | Hadoop Experts | Cazton". In: (). URL: https://cazton.com/consulting/big-data-development/apache-hadoop-and-hdfs (cit. on p. 6).

[38] I. A. T. Hashem et al. "The rise of "big data"on cloud computing: Review and open research issues". In: *Information Systems* 47 (2015), pp. 98–115. ISSN: 03064379. DOI: 10.1016/j.is.2014.07.006. URL: http://dx.doi.org/10.1016/j.is.2014.07.006 (cit. on p. 13).

[39] D. Haz. *ARC: React starter kit based on Atomic Design*. https://github.com/diegohaz/arc/wiki. May 2017 (cit. on p. 33).

[40] A. Holst. *A day in Data*. TotalDataVolumeWorldwide2010-2025. June 2021 (cit. on p. 1).

[41] Y. Hu and G. Bai. "A Systematic Literature Review of Cloud Computing in Ehealth". In: *Health Informatics - An International Journal* 3.4 (2014), pp. 11–20. ISSN: 23193190. DOI: 10.5121/hiij.2014.3402. arXiv: 1412.2494 (cit. on p. 8).

[42] W. Inoubli et al. "An experimental survey on big data frameworks". In: *Future Generation Computer Systems* 86 (2018), pp. 546–564. ISSN: 0167739X. DOI: 10.1016/j.future.2018.04.032. arXiv: 1610.09962. URL: https://doi.org/10.1016/j.future.2018.04.032 (cit. on p. 6).

[43] *Introduction | Jaspersoft Community*. https://community.jaspersoft.com/documentation/tibco-jasperreports-io-user-guide/v790/introduction. (Accessed on 09/18/2021) (cit. on p. 17).

[44] S. R. Khan. "MSL Framework: Minimum Service Level Framework for Cloud Providers and Users TT - MSL Framework: Framework de Nível de Serviço Mínimo para Provedores e Usuários de Nuvem". In: *PQDT - Global* (2018), p. 401. URL: https://search.proquest.com/docview/2297419480?accountid=13460%0Ahttp://zp2yn2et6f.search.serialssolutions.com/directLink?&atitle=MSL+Framework%3A+Minimum+Service+Level+Framework+for+Cloud+Providers+and+Users&author=Khan%2C+Sohail+Razi&issn=&title=MSL+Fra (cit. on pp. 8, 9).

[45] C. S. Kruse et al. "Challenges and opportunities of big data in health care: A systematic review". In: *JMIR Medical Informatics* 4.4 (2016), pp. 1–11. ISSN: 22919694. DOI: 10.2196/medinform.5359 (cit. on p. 1).

[46] T. A. Majchrzak, T. Jansen, and H. Kuchen. "Efficiency evaluation of open source ETL tools". In: *Proceedings of the ACM Symposium on Applied Computing* (2011), pp. 287–294. DOI: 10.1145/1982185.1982251 (cit. on p. 16).

[47] M. Malawski. "Towards serverless execution of scientific workflows - HyperFlow case study". In: *CEUR Workshop Proceedings* 1800 (2016), pp. 25–33. ISSN: 16130073 (cit. on p. 12).

[48] M. Malawski et al. "Serverless execution of scientific workflows: Experiments with HyperFlow, AWS Lambda and Google Cloud Functions". In: *Future Generation Computer Systems* 110 (2020), pp. 502–514. ISSN: 0167739X. DOI: 10.1016/j.future.2017.10.029. URL: https://doi.org/10.1016/j.future.2017.10.029 (cit. on pp. 10, 12).

[49] N. Mali. "A Survey of ETL Tools". In: *International Journal of Computer Techniques* -– 2.5 (2015), pp. 20–27. ISSN: 2394-2231. URL: http://www.ijctjournal.org (cit. on pp. 15, 16).

[50] J. Manyika et al. "Big data: The next frontier for innovation, competition and productivity". In: *McKinsey Global Institute* June (2011), p. 156. URL: https://bigdatawg.nist.gov/pdf/MGI_big_data_full_report.pdf (cit. on pp. 2, 4).

[51] C. McAnlis. *Improving Cloud Function cold start time*. https://medium.com/@duhroach/improving-cloud-function-cold-start-time-2eb6f5700f6. Mar. 2018 (cit. on p. 56).

[52] F. Ødegårdstuen. *How to increase Google Cloud Function startup speed tenfold*. https://medium.com/@Frikkylikeme/how-i-fixed-slow-function-startups-in-google-cloud-a7482056a415. June 2020 (cit. on p. 56).

[53] S. Overflow. *Cloud SQL features | Cloud SQL Documentation | Google Cloud*. https://insights.stackoverflow.com/survey/2021#technology-most-popular-technologies. June 2021 (cit. on p. 38).

[54] S. Overflow. *gcloud functions | Cloud SDK Documentation | Google Cloud*. https://cloud.google.com/sdk/gcloud/reference/functions/. June 2021 (cit. on p. 42).

[55] V. M. Parra and M. N. Halgamuge. "Performance Evaluation of Big Data and Business Intelligence Open Source Tools: Pentaho and Jaspersoft". In: (2018), pp. 147–176. DOI: 10.1007/978-3-319-60435-0_6 (cit. on p. 16).

[56] J. Peng et al. "Comparison of several cloud computing platforms". In: *2nd International Symposium on Information Science and Engineering, ISISE 2009* (2009), pp. 23–27. DOI: 10.1109/ISISE.2009.94 (cit. on p. 8).

[57] *Pentaho Data Integration - Hitachi Vantara Lumada and Pentaho Documentation*. https://help.hitachivantara.com/Documentation/Pentaho/7.1/0D0/Pentaho_Data_Integration#Using_the_PDI_Client. (Accessed on 09/18/2021) (cit. on p. 17).

[58] E. Pulsifer and M. Andersson. *What is google cloud platform (GCP)?* https://acloudguru.com/blog/engineering/what-is-google-cloud-platform-gcp. Nov. 2021 (cit. on p. 30).

[59] M. Reisman. "EHRs: The challenge of making electronic data usable and interoperable". In: *P and T* 42.9 (2017), pp. 572–575. ISSN: 10521372 (cit. on p. 5).

[60] J. Roski, G. W. Bo-Linn, and T. A. Andrews. "Creating value in health care through big data: Opportunities and policy implications". In: *Health Affairs* 33.7 (2014), pp. 1115–1122. ISSN: 15445208. DOI: 10.1377/hlthaff.2014.0147 (cit. on p. 2).

[61] S. SA. "Big Data in Healthcare Management: A Review of Literature". In: *American Journal of Theoretical and Applied Business* 4.2 (2018), p. 57. ISSN: 2469-7834. DOI: 10.11648/j.ajtab.20180402.14 (cit. on pp. 2, 3, 6, 13–15, 18).

[62] N. Schmidt et al. "ETL Tool Evaluation-A Criteria Framework". In: *Southwest Decision Sciences Insitute* 956 (2011), p. 12. URL: http://swdsi.org/swdsi2011/2011_SWDSI_Proceedings/papers/papers/PA189.pdf (cit. on pp. 15, 16).

[63] SeattleDataGuy. *What Are ETLs and Why Are They Important? | by SeattleDataGuy | Better Programming*. https://betterprogramming.pub/what-are-etls-and-why-are-important-b65b301607d1. (Accessed on 09/11/2021) (cit. on p. 15).

[64] C. Thomsen and T. B. Pedersen. "A survey of open source tools for Business Intelligence". In: *International Journal of Data Warehousing and Mining* 5.3 (2009), pp. 56–75. ISSN: 15483932. DOI: 10.4018/jdwm.2009070103 (cit. on p. 16).

[65] TIBCO Software Inc. *Jaspersoft ETL ™ - Data Integration for BI*. URL: https://www.jaspersoft.com/sites/jaspersoft/files/ds-jaspersoft-etl-final-fy19.pdf (visited on 09/18/2021) (cit. on p. 17).

[66] *Ultimate Cloud Pricing Comparison: AWS vs. Azure vs. Google Cloud in 2021 - CAST AI*. https://cast.ai/blog/ultimate-cloud-pricing-comparison-aws-vs-azure-vs-google-cloud-in-2021/. (Accessed on 09/10/2021). Aug. 2021 (cit. on p. 9).

[67] H. Vantara. *Lumada DataOps Suite : Pentaho Data Integration*. URL: https://www.hitachivantara.com/en-us/pdfd/datasheet/pentaho-business-analytics-platform-datasheet.pdf (visited on 09/18/2021) (cit. on p. 17).

[68] A. Verma. *What is the role of src and dist folders in JavaScript/jquery?* https://cloud.google.com/appengine/docs/standard/nodejs/config/appref. Apr. 2020 (cit. on p. 33).

[69] *What is Data Governance? | Google Cloud*. https://cloud.google.com/learn/what-is-data-governance. (Accessed on 09/11/2021) (cit. on p. 15).

[70] A. Wirfs-Brock and B. Eich. "JavaScript: The first 20 years". In: *Proceedings of the ACM on Programming Languages* 4.HOPL (2020). ISSN: 24751421. DOI: 10.1145/3386327 (cit. on p. 18).

I

Pentaho ETL Tool Transformations

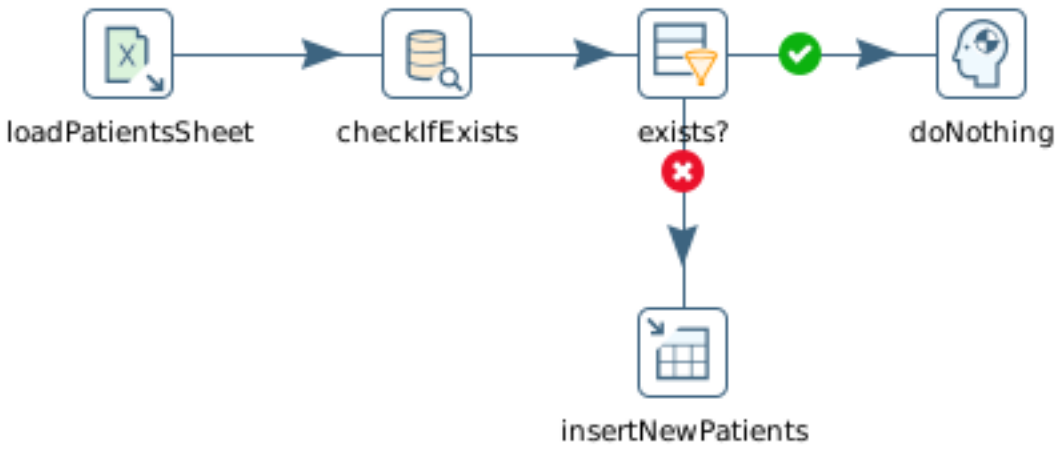


Figure I.1: Patients table transformation.

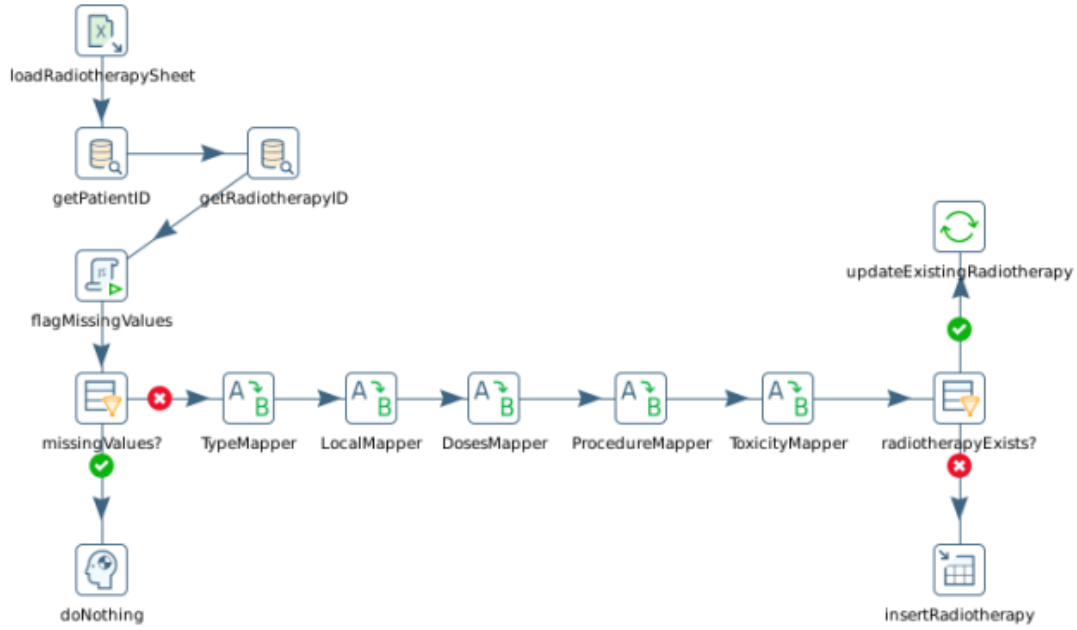


Figure I.2: Radiotherapy table transformation.

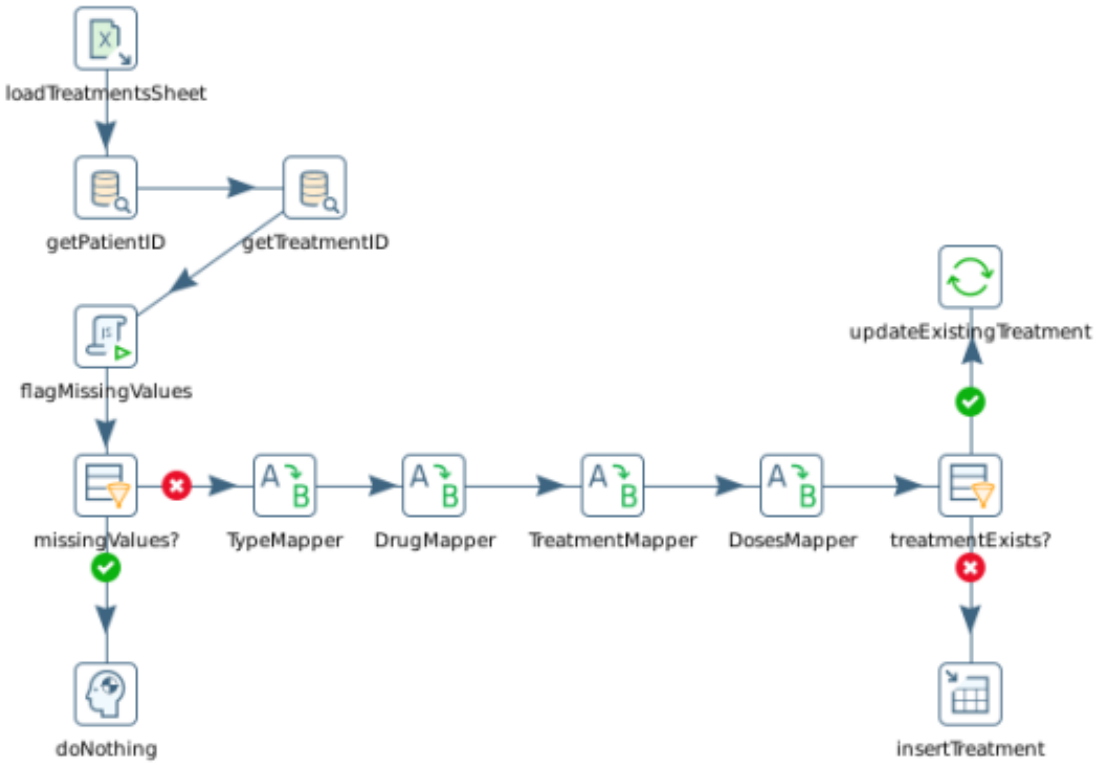


Figure I.3: Treatments table transformation.

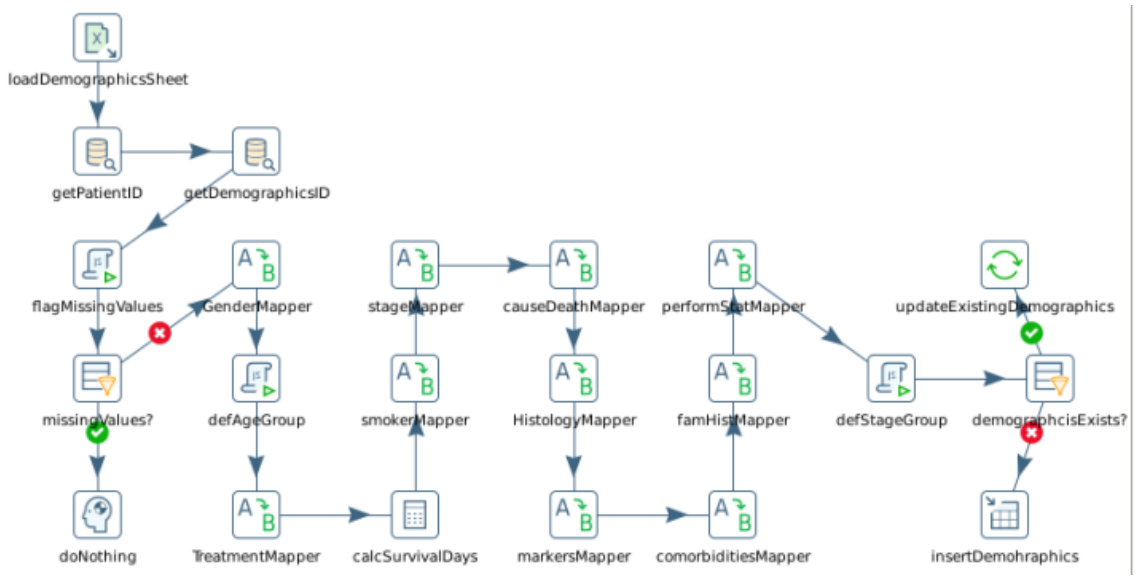


Figure I.4: Treatments table transformation.

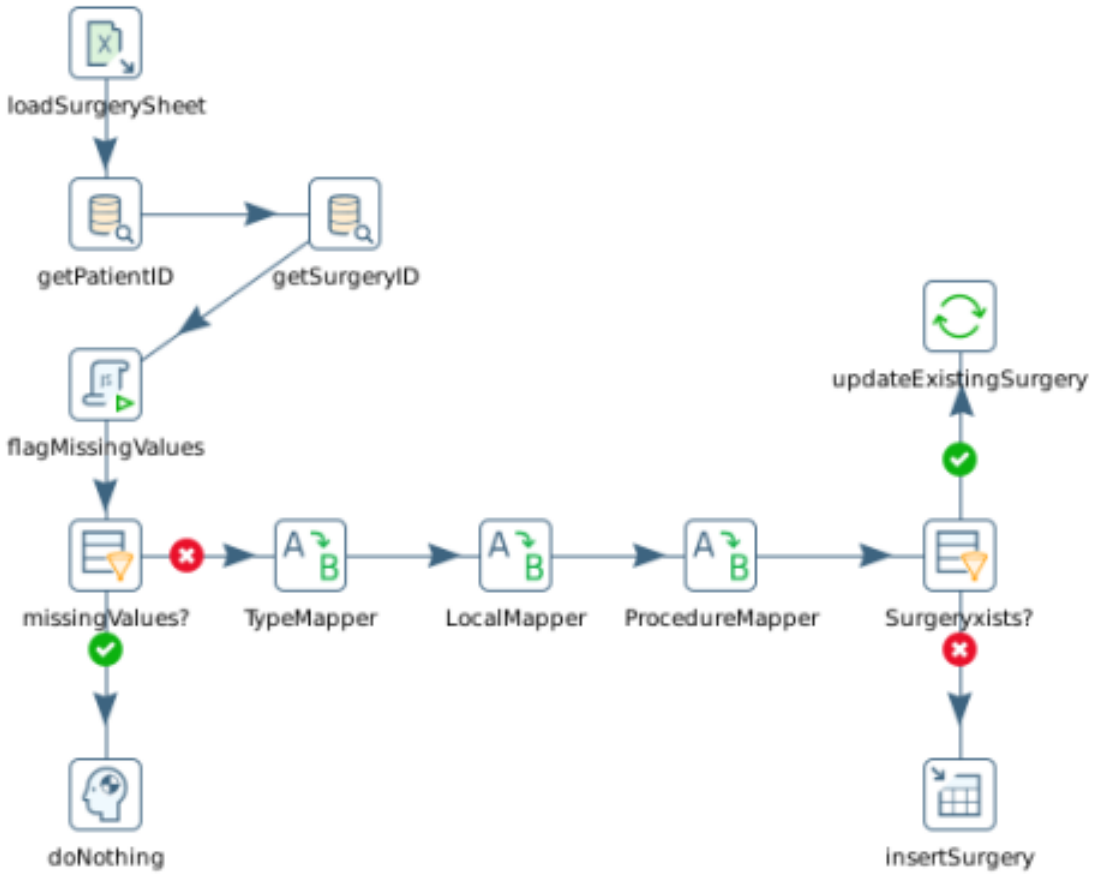


Figure I.5: Surgery table transformation.

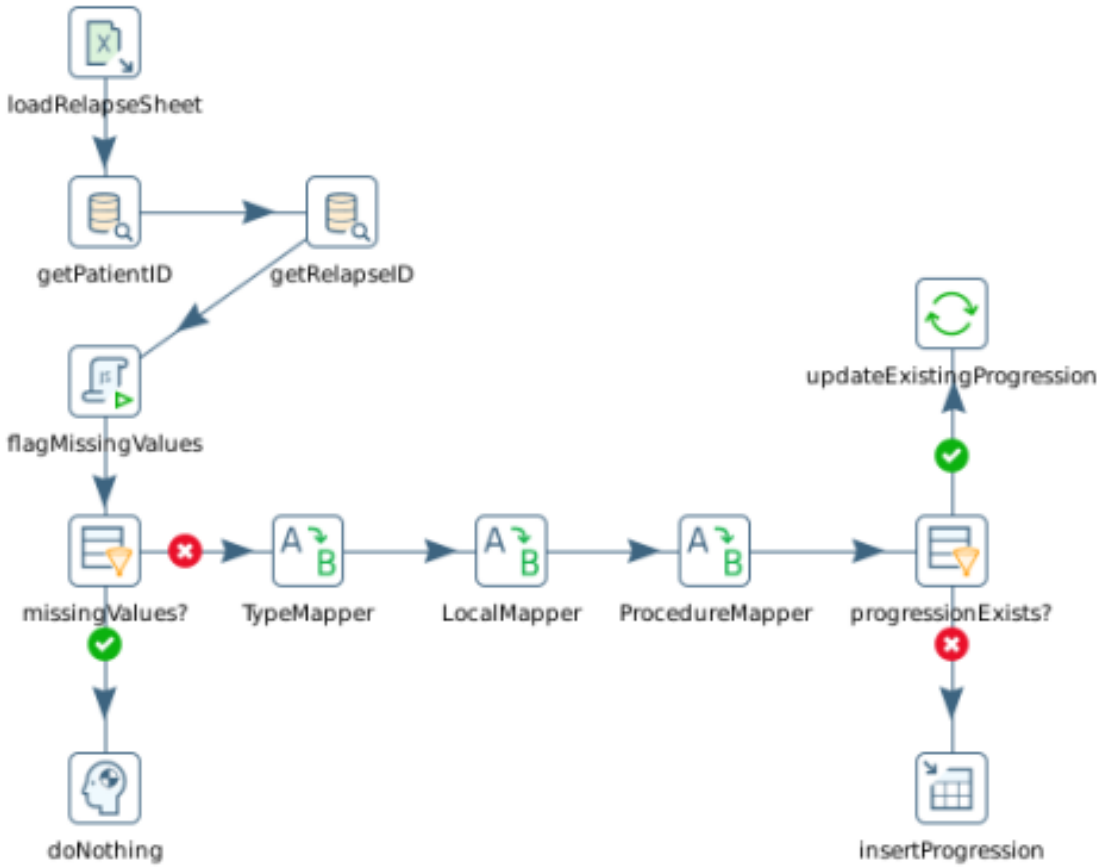


Figure I.6: Progression table transformation.

II

Google Cloud Functions Optimisation Results

#	Cold Start (ms)			#	Cold Start Optimized (ms)		
1	4287			1	3829		
2	3929			2	3823		
3	4579			3	2621		
4	3727			4	2797		
5	3389			5	3114		
6	3470			6	2660		
7	3487			7	2912		
8	3715			8	2048		
9	3671			9	3106		
10	3573			10	2546		
av	3782.7	stdev	382.1244852	av	2945.6	stdev	555.7599802

Figure II.1: dataset_info cold start optimisation results.

#	Warm response (ms)			#	Warm response Optimized(ms)		
1	1218			1	205		
2	1211			2	107		
3	1232			3	465		
4	1244			4	383		
5	1605			5	90		
6	1188			6	103		
7	1247			7	189		
8	1234			8	294		
9	1259			9	363		
10	1347			10	237		
av	1278.5	stdev	122.2049735	av	243.6	stdev	129.8642026

Figure II.2: dataset_info warm response optimisation results.

#	Cold Start (ms)			#	Cold Start Optimized (ms)		
1	16389			1	922		
2	12207			2	1178		
3	13164			3	852		
4	15608			4	768		
5	20643			5	1006		
6	18261			6	1183		
7	13303			7	907		
8	13508			8	1183		
9	13650			9	942		
10	17272			10	1061		
av	15400.5	stdev	2716.915214	av	1000.2	stdev	147.664635

Figure II.3: patient_info cold start optimisation results.

#	Warm response (ms)			#	Warm response Optimized(ms)		
1	3675			1	445		
2	3633			2	316		
3	4497			3	413		
4	4493			4	668		
5	3585			5	561		
6	1619			6	765		
7	4320			7	579		
8	2249			8	412		
9	3035			9	647		
10	3828			10	666		
av	3493.4	stdev	950.7033887	av	547.2	stdev	144.3343341

Figure II.4: patient_info warm response optimisation results.

#	Cold Start (ms)			#	Cold Start Optimized (ms)		
1	14599			1	1662		
2	21584			2	1620		
3	15397			3	2099		
4	12926			4	2001		
5	13007			5	1625		
6	11928			6	1897		
7	12207			7	1923		
8	12195			8	1802		
9	18251			9	2147		
10	12678			10	1940		
av	14477.20	stdev	3167.097087	av	1871.6	stdev	190.074956

Figure II.5: survival_curve cold start optimisation results.

#	Warm response (ms)			#	Warm response Optimized(ms)		
1	8165			1	1166		
2	8697			2	1303		
3	8533			3	1204		
4	8233			4	1316		
5	8044			5	1318		
6	8140			6	1193		
7	8371			7	1180		
8	7952			8	1065		
9	7997			9	1337		
10	8611			10	1195		
av	8274.3	stdev	265.3756457	av	1227.7	stdev	87.53164507

Figure II.6: survival_curve warm response optimisation results.