isec
Engenharia

MESTRADO EM INFORMÁTICA E SISTEMAS

Real-Time Data Analytic Platform

Autor

Dany Fajardo Carvalho

Orientadores

José Marinho

Coimbra, março 2021

DEFINITIVO

INSTITUTO POLITÉCNICO
DE COIMBRA

INSTITUTO SUPERIOR
DE ENGENHARIA
DE COIMBRA

DEPARTAMENTO DE INFORMÁTICA E SISTEMAS

Real-Time Data Analytic Platform

Relatório de Estágio de Natureza Profissional para a obtenção do grau de Mestre em Informática e Sistemas

Especialização em Desenvolvimento de Software

Autor

Dany Fajardo Carvalho

Orientadores

José Marinho

Supervisores na empresa Critical Software S.A.

Álvaro Menezes

Pedro Miranda

Coimbra, março 2021

INSTITUTO POLITÉCNICO DE COIMBRA

INSTITUTO SUPERIOR DE ENGENHARIA DE COIMBRA

This dissertation is dedicated to my family.

Acknowledgements

Even before we dive into the whole document there is always time to thank all the people that contributed to this project. The remainder of this section is therefore more intimate and presents some Portuguese words so that, at least, something can be understood by familiar members.

Antes de mais, gostaria de agradecer aos meus pais, José Carvalho e Olinda Fajardo, por me terem apoiado em todo o meu percurso académico, assim como durante toda a minha vida, e por estarem sempre dispostos a ajudar em o que quer que seja.

Ao meu irmão, Fábio Fajardo, que apesar da distância que nos separa entre países, sempre me viu com olhos de orgulho e esteve pronto com palavras de incentivo, assim como a sua esposa e filha, minha afilhada.

Já mais por perto, gostaria de agradecer ao meu namorado, Telmo Julião, por todo o conforto e carinho que me deu durante todo este percurso de estágio, incluindo os bons momentos que me proporcionou durante o confinamento e fora disso.

É com grande satisfação que agradeço também ao meu orientador, Doutor José Marinho, docente no Instituto Superior de Engenharia de Coimbra, por toda a compreensão que teve durante este processo de estágio, assim como todos os conhecimentos transmitidos, incluindo a ajuda na redação deste relatório.

Também envio um enorme abraço aos engenheiros Álvaro Menezes e Pedro Miranda, da Critical Software, que estiveram a certa altura presentes a acompanhar-me neste projeto académico e que sempre me deram conselhos bastante importantes.

Aos restantes familiares, amigos e colegas de faculdade e de trabalho, queria deixar uma palavra de agradecimento, sendo que foram um incentivo para eu não desistir.

Finalmente, gostaria de enviar as melhores recuperações aos meus pais, que infelizmente, em 2020 não tiveram a melhor sorte, tendo sido detetado o cancro da mama à minha mãe e o meu pai ter sofrido um severo Acidente Vascular Cerebral. Aos dois, as profundas melhoras e que nunca desistam de lutar na vida e por quem mais vos ama. Isto é por vocês.

Obrigado – Thank you.

Abstract

The world of data has increased in today's data science and data engineering fields. Data analytics, from a perspective, is increasing since they are useful to get insights of a company and guarantee an excellent business opportunity precisely because of data which are more present due to Artificial Intelligence, Internet of Things (IoT), social media and software/hardware components.

In order to process, analyze and deliver data in a very short period of time, streaming gained its form, and real-time data analytic platforms began to rise, leaving the traditional batching process aside. In fact, to build a data analytics platform, real-time or not, Big Data joins in and provides their architectures with their components.

Lambda and Kappa, two existing Big Data architectures, consist of numerous components, i.e., supporting 'blocks', offering the opportunity to explore their functionalities to develop real-time data analytic platforms. When implementing such solutions, another question that might arise is the choice of which architecture might best fit the targeted business type.

This internship report focuses on analyzing and drawing some conclusions about any possible correlation between business types and specific data analytics solutions that are better suited to support them. It also discusses if creating a generic real-time data analytic platform (i.e., appliable to any kind of existing business type) is feasible, which could significantly decrease development and deployment costs. Specifically, Lambda and Kappa are inspected in this report to understand if these Big Data architectures are generic enough to address this issue or if a customization based on their components is possible. Proving that any of the analyzed Big Data architectures are feasible to be implemented to any real-time data analytic platform, the remaining report addresses the development of a specific use case covering Kappa as its main architecture.

Resumo

Atualmente, o mundo dos dados está a crescer, sobretudo nas áreas de *Data Science* e *Data Engineering*. A análise de dados, tem-se tornado cada vez mais relevante para obter um conhecimento mais profundo sobre uma determinada empresa e representa uma oportunidade de negócio, precisamente devido à emergente presença de dados derivados da Inteligência Artificial, *Internet of Things* (IoT), *social media* e componentes de software/hardware.

De modo a processar, analisar e distribuir estes dados num curto espaço de tempo, o *streaming* tem ganho popularidade e as plataformas de análise de dados em tempo real começaram a surgir, colocando de lado os tradicionais processamentos de dados por lotes. De facto, para desenvolver uma plataforma de análise de dados, em tempo real ou não, as arquiteturas de *Big Data* e os seus componentes tornaram-se essenciais.

As arquiteturas de *Big Data* existentes, Lambda e Kappa, são suportadas por vários componentes, oferecendo a oportunidade de explorar as suas funcionalidades para desenvolver plataformas de análise de dados em tempo real. Ao implementar este tipo de soluções, surge, por vezes, a questão sob qual das arquiteturas será a mais adequada a um determinado tipo de negócio.

Neste relatório de estágio, é demonstrada a análise e conclusões sobre uma possível correlação entre os tipos de negócio e quais as soluções de análise de dados mais adequadas para os suportar. Ao longo deste documento, é ainda ponderada a possibilidade de desenvolver uma plataforma de análise de dados em tempo real, genérica o suficiente, para ser aplicável em qualquer tipo de negócio, reduzindo significativamente os custos de desenvolvimento e implementação. Neste contexto, são examinadas as arquiteturas Lambda e Kappa, por forma a entender se são suficientemente universais para essa possibilidade ou se é viável uma personalização baseada nos seus componentes. De modo a comprovar se qualquer uma destas arquiteturas de *Big Data* é implementável numa plataforma genérica de análise de dados em tempo real, o relatório também descreve o desenvolvimento de um caso de uso específico baseado na arquitetura Kappa.

Table of Contents

List of Figures

List of Tables

Acronyms and Definitions

A

AI
Artificial Intelligence 2

C

CEO
Chief Executive Officer 1
CSW
Critical Software 1

D

DA
Data Analytics 2

E

ETL
Extract-Transformation-Load 16

I

IDE
Integrated Development Environment 23
IoT
Internet of Things .... ix, 2
IP
Internet Protocol 23
ISEC
Instituto Superior de Engenharia de Coimbra 1

J

JSON
JavaScript Object Notation 33

M

MIS
Mestrado em Informática e Sistemas 1
ML
Machine Learning 9

1 Introduction

This report describes the work performed during an internship integrated in the Master's course in Informatics and Systems (*Mestrado em Informática e Sistemas* – MIS) at the *Instituto Superior de Engenharia de Coimbra* (ISEC) [1] for the academic year 2019/2020. The duration of this internship was two semesters and has been distributed along two academic semesters, starting on November the 14th, 2019, at Critical Software [2] headquarters in Coimbra and then later being transferred voluntarily to Lisbon Offices for the second semester, in February 2020. Its goal was to develop a real-time data analytic platform, as generic as possible, to be applied to any business type case and since Critical software did not have any similar project that followed that approach, the development of this project was one big accomplishment.

In the following sections, Critical Software and ISEC are presented, a more contextualized scope and background is also provided, and the structure of the report follows in the last section.

1.1 Critical Software

Critical Software, internally referenced as CSW, is an internationally respected and recognized organization which started from the bottom but were right on top of systematic excellent work delivery. All started when Diamantino Costa, Gonçalo Quadros, and João Carreira, now the actual CEO (Chief Executive Officer), wrote several papers about the space sector and consequently attracted NASA (National Aeronautics and Space Administration) in 1998, year in which Critical Software was born.

Afterwards they were unstoppable and managed to gather international success such as being chosen as a case study by The European Space Agency (ESA). Also, they were the first Iberian company to receive the ISO9001:2000 TickIT quality certification. They contributed to ESA's EarthCARE space mission project, they launched several spin-offs and are one of the few companies in the world achieving the CMMI-SE/SW Level 5 certification [3].

Their work was being undoubtedly recognized and the dream of expanding became reality, having currently 9 offices distributed around the globe. The space sector is not the only industry CSW explores but also Aerospace, Automotive, Defence, Energy, Finance, Government, Medical Devices, Railway, Space and Telecoms. In each sector, CSW works on '*demanding projects, providing software and system services for safety, mission, and business-critical applications*' [4], ensuring high quality throughout the entire project.

1.2 ISEC

Instituto Superior de Engenharia de Coimbra, or ISEC, is an Institute of Engineering located in Coimbra and offers several Bachelor and Master courses, as well as other professional courses in different areas of engineering, such as Chemical, Civil, Electrical, Information

Technology, and Mechanical. Moreover, ISEC '*has as its mission the creation, transmission and diffusion of culture, science and technology, being responsible for providing a higher education for the exercise of professional activities in the field of Engineering and promoting the development of the region*' [5].

Finally, and extremely important to increase students' passion in creating innovative projects, ISEC encourages them to participate in the Fikalab [6] project. Through the participation, not only are students able to win special nominations and prizes but also experience in developing fun and unique projects, individually or in a group. This initiative is also supported by CSW and the students' involvement increases their capacity of developing in a fun and healthy way and to bring some challenges to their professional lives.

1.3 Context and Problem Statement

Data nowadays are undoubtedly as important as the ignition of a rocket, carbon for an old locomotive, electricity for a smart gadget, or even coffee for our brains. This increasing importance is due to a huge evolution in our technology era, which results in data volumes being expected to double every two years ("Fast Data" designation [7]). The evolution of Artificial Intelligence (AI), Internet of Things (IoT), Cloud Computing, and Edge Computing largely contribute to this phenomenon [7].

Organizations take Data very seriously since it is one of the most valuable resources for their business models. There are even companies that make money selling personal information to other organizations [8]. Transformation and aggregation operations must be applied to data prior allowing the company analysts to come up with business insights and future opportunities. Analyzing collected data and giving some useful insights to the company is one of the components of Data science, more precisely Data Analytics (DA). The maintenance of data, the exploration of historical overview records, and the availability of real-time data are three key factors that enable providing companies with solutions that help them predicting and making decisions.

Big Data is a well-recognized area in the world of Data and the use of open-source tools has been considered to 'address the storage, processing and visualization of such data' [9]. Big Data is a collection of large datasets which are complex to work with when using traditional methods like regular databases. Also, Big Data is well-known for their 5 V (Volume, Variety, Velocity, Value, and Veracity) and for their architectures.

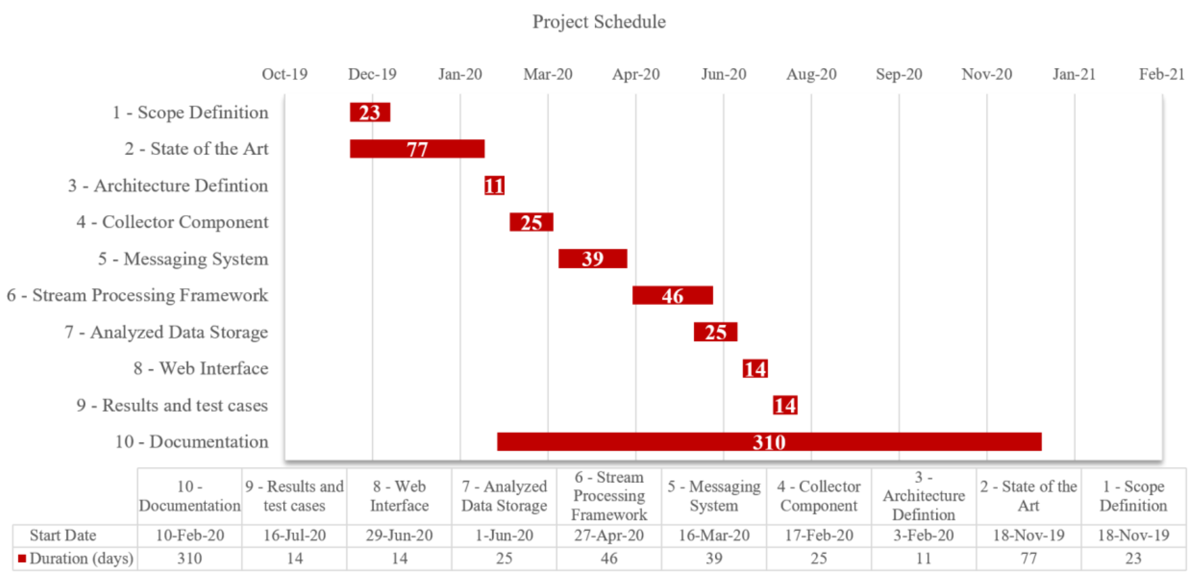
In this context, this report aims to analyze existing state of the art Big Data solutions, namely Lambda [10] and Kappa [10], and also to determine if they are generic enough such that they can be applied to any business type. It is organized in various chapters and starts providing some background information about key concepts, such as data analytic and big data. Then, it analyses related work about Big Data architectures, covering Lambda and Kappa architecture in greater detail, demonstrating how they are built and specifying how they differ from each other. Furthermore, two case studies are presented and compared, so that conclusions could be conducted either which of the Big Data architectures could be suitable to any type of business cases. Both presented case studies follow a distinct Big Data architecture so that the advantages and the disadvantages are clearly understood, emphasizing the choice of Lambda and Kappa architectures within each different business type systems. In addition, this project's scope is to develop a real-time data analytic platform which needs an infrastructure that supports and therefore, the chosen architecture is based on the comparison of the Lambda and Kappa architecture previously mentioned.

1.4 Workplan and Methodology

At an initial stage of the internship, it was crucial to start with some basics and background information within this project's scope. Since it was not within the specialization domain of the trainee's Master course, it was clearly important to invest more in Data Analytics[1] and Big Data[2] areas. Therefore, a literature review covering some important background concepts, such as Big Data, Data Analytics, Batch, and Streaming processing, was performed. Also, the CSW's internship proposal (Appendix A) presented a high-level pipeline projection that contributed to an initial input that could be certified while searching for Big Data architectures. In this context, and even in a brighter spectrum like Data Science, a pipeline can be compared to a process that gets input data, that also can perform transformations during the process time range (e.g.: inside the 'pipe') and end up with an output result. The pipeline projection in Appendix A is composed by several components and those are within Lambda or Kappa architecture.

The Gantt chart in Figure 1-1 lists all the tasks that were performed, such as the literature review, as well as other significant tasks that contributed to the implementation of a real-time data analytic platform pipeline. Also, the effort reported for every task is measured in days and these effort values are displayed in the table within Figure 1-1. Regarding the documentation phase, its duration is relatively higher than the other tasks because of some limitations that are described in section 6.1 and, the fact that this is not a continuous process.

[1] Science of examining data to make conclusions.
[2] Area in which a huge amount of data is transformed, processed, and worked with.



	10 - Documentation	9 - Results and test cases	8 - Web Interface	7 - Analyzed Data Storage	6 - Stream Processing Framework	5 - Messaging System	4 - Collector Component	3 - Architecture Defintion	2 - State of the Art	1 - Scope Definition
Start Date	10-Feb-20	16-Jul-20	29-Jun-20	1-Jun-20	27-Apr-20	16-Mar-20	17-Feb-20	3-Feb-20	18-Nov-19	18-Nov-19
🟥 Duration (days)	310	14	14	25	46	39	25	11	77	23

Figure 1-1: Workplan in Gantt chart

Since the beginning, the steps to carry this project along were clearly defined by CSW, and the proposal itself stated to follow a Waterfall methodology. Due to the global pandemic, the project started to change its course. In March, it began to suffer some adjustments, more meetings were being held, other requirements were added, and some tailoring was conducted, among other tasks that are mentioned in Chapter 4.

The meetings were held once a week via Microsoft Teams platform. Initially, the assigned CSW supervisor was Álvaro Menezes who was replaced afterwards by the current project supervisor Pedro Miranda. Also, to mention that Professor José Marinho from ISEC was one of the main supervisors of this internship, and several meetings were held with him to monitor the course of the project development and report writing.

1.5 Outline

The remainder of this report is organized as follows:

Chapter 2: Literature review. The origins of Big Data and Data Analytics concepts, their scope, a background section, and an overview of the literature on these subjects are provided. An analysis of existing state of the art solutions and two use cases are also presented.

Chapter 3: Requirement analysis and architecture design. All the functional and non-functional requirements are listed according to the development needs. The architecture of the developed solution is specified along with necessary tools and frameworks.

Chapter 4: Implementation. This chapter contains all the implementation choices throughout the entire project development phase, as well as a description of the tools and technologies that were used.

Chapter 5: Results and Test Cases. In this chapter the result of the development phase and the test cases are presented.

Chapter 6: Conclusions and Future Research directions. This chapter draws final conclusions about the entire study and project development. Furthermore, it also includes possible future developments.

2 Literature review

This chapter covers some background topics to understand the projects' scope. Some topics related to Big Data, Data Analytics, as well as batch and stream processing principles are described in the related work section. Lambda and Kappa are also clearly identified as main architectural models to support a real-time data analytic platform infrastructure. These models are based on individual components and those can be implemented by using several technologies. Along with the Lambda and Kappa specifications, two different business cases are presented to distinguish the use of each architecture according to the respective use case.

2.1 Background

Even though this report is about Data Analytics, it is important to highlight the importance of Big Data in this context. Data analytics are supported by Big Data architectures. Just cruising back to 2012 when the term Big Data was no longer a secret for its use and mission, the three V were presented as the answer to treat large datasets (Volume) from different data sources (Variety), in a high frequency of time to collect and publish this information (Velocity) [11]. Its existence is due to today's huge amount of data, which turns traditional database systems unable to process those large datasets. Additionally, data come from distinct data sources in different forms, like structured, unstructured, and semi-structured formats. Since data is growing, Gartner (former MetaGroup analytic firm) [12] added two more relevant V (Value and Veracity) aiming for providing value to the company and quality to the collected data [12].

Considering the main reasons why Big Data is known for and its main characteristics, Data Analytics is in some way connected to it since it is a process ahead, i.e., the science of examining those collected data and giving some useful insights to the company. However, just collecting information does not mean that it will correspond to good results. The right information must be collected and then translated into some profitable result, in the right place, and at the right time in a near future. Companies want results as fast as they can and while they are still up to date and useful to them, i.e., in real-time.

Creating platforms for working with big volumes of data and getting some analytics about them in a hopefully short and useful period is not completely achievable through batch processing, which consists in fetching some analytics sometime after data collection. This approach, which continues being applied nowadays, is not appropriate to achieve consumer preferences, satisfaction, company insights, and fraud detection in a new era where streaming and results are important factors [13]. Thus, the need to process results in real time resulted in the development of a technology designated as streaming processing. While batch processing consists in collecting data over time and fetching some analytics later after, especially overnight, stream processing, on the other hand, consists in collecting data as soon as it is available and delivering results in real-time or, more precisely, in near real-time [10].

2.2 Related Work

After some background about Big Data, Data Analytics, and Batch and Stream processing provided in the previous section, this section describes in more detail these topics and demonstrates the connection between them. As a starting point, a generic Big Data diagram that consists of different components is considered (Figure 2-1). Then, the Lambda (λ) and Kappa (κ) Big Data architectures, which include some of these components, are described in greater detail with their main characteristics, differences, advantages, and disadvantages. For this project, only Lambda and Kappa were considered to be analyzed since other architectures, such as Unified Lambda architecture, Mu architecture and Zeta architecture, are variations of the Lambda and Kappa architectures [9]. Moreover, two different business types are described following different architecture approaches, namely the Lambda architecture for a streaming service and the Kappa architecture for a Telco company.

2.2.1 Big Data Architectures and their components

Figure 2-1 [14] is a generic pipeline that includes the components of both Big Data architectures, Lambda and Kappa, but also the variations previously mentioned. Not every component is mandatory while building a Data Analytics platform and the goal of this section is to describe each one.

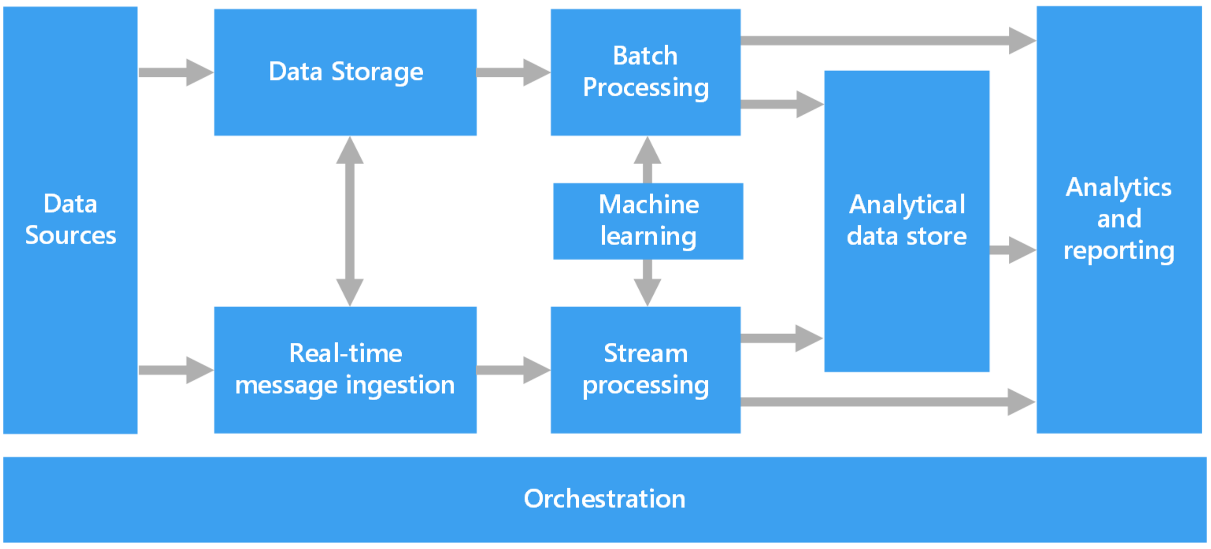


Figure 2-1: Big Data Components diagram [14]

Data come across from diverse data sources in distinct data formats. Structured, unstructured, and semi structured formats are the most common ones. Examples of structured data are relational databases and spreadsheets. Unstructured data examples are e-mails, portable document format (PDF) files, images, and videos. The semi-structured data format is visible in extensible Markup Language (XML) files or Comma-Separated Values (CSV) documents. It is basically information that is not allocated in relational databases but has attributes that help in the analysis process.

These incoming data can be stored in the Data Storage component, which is usually a Data Lake[3] or a Data Warehouse[4]. These systems maintain the data in its raw format and help when it comes to batch and stream processing.

Even though real-time message ingestion, stream processing, and batch processing components are presented individually on Figure 2-1, they can be implemented as a single component through Apache Kafka [15] which is a powerful streaming processing platform that leads with different data in near real-time. It is based on the Producer and Consumer pattern, where the producer publishes log messages to a Kafka Topic, also known as Kafka broker. Each message is allocated to a partition and is set with an offset, while the consumer subscribes to a topic and consumes all the messages that are within the partition(s).

Machine Learning (ML) algorithms can be implemented to identify any related pattern and to train data to discover some useful information. In this category, two classes of algorithms can be used, namely Supervised and Unsupervised. Supervised algorithms have access to the input and output data, such as Classification models (Naïve Bayes, SVM) and Regression (Linear, Logistic), while unsupervised algorithms do not have access to the outputs in advance, working only with the input data. Clustering (K Means) and Dimensionality Reduction (Principal Component Analysis, SVD) are examples of unsupervised algorithms [16] [17].

The analytical data store component can be the representation of the result views. Queries are made through those views and the information is returned to the end-user. Analytics and reporting are the data visualization part where any result is presented on a dashboard or web interface.

Considering the two currently most used Big Data architectures, Lambda and Kappa, the next two sections will cover their usage and demonstrate how each architecture differs from one another. To mention that, both Lambda and Kappa can have similar technologies in each further described component [9]. As for a developer, the development of the component does require the use of an existing tool or framework. In other words, similar technologies and frameworks like Apache Kafka, Apache Spark can figure in these architectures but can be implemented using different programming languages. For instance, Apache Spark allows us to develop on Java, Scala or even Python programming language [16].

2.2.2 Lambda Architecture

Lambda architecture (Figure 2-2) has been named by Nathan Marz [10]. Back in 2011, the former lead engineer of the company BackType [18], stated this architecture as a '*generic, scalable and fault-tolerant real-time data processing architecture*' [10]. Even before streaming was possible, big companies had to process their data by batch processing. This mode of operation requires too much time, leading to a high latency and therefore to an overnight task. By observing Figure 2-2, and knowing that batching is a task that carries a huge latency, it may seem controversial to see the batching process present, making it less probable that this architecture is even suitable for real-time streaming. Fortunately, Lambda does include the

[3] Repository for storing structured, unstructured, and semi-structured in its raw format
[4] System that stores data along with metrics, expected to be processed by an ETL process

speed layer, which fixes the latency problem. So, the high latency problem is fixed by assuming pre-computed views (i.e., data that the batch layer processes) and indexing them with the low latency real-time views (i.e., current data from the speed layer) to a merged view results, namely Serving layer. Back then, this architecture seemed to be a successful architecture to support real-time analytics with low latency.

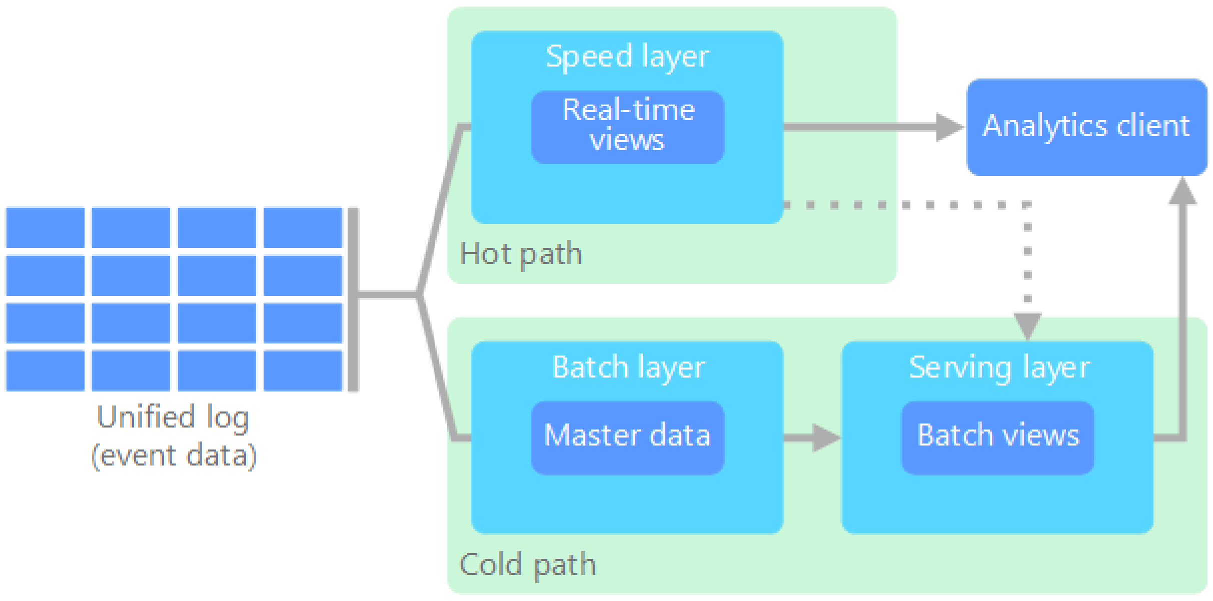


Figure 2-2: Lambda's Architecture [14]

Marz wanted to beat the CAP (Consistency, Availability, Partition) Theorem [19], which in his beliefs this architecture could master. Ultimately, his architecture does not resolve the CAP theorem and this is also proved in article [10] stated by the authors that '*the architecture does not rebut the CAP theorem but simplifies its [architecture] complexity…*', which is also criticized by the IEEE Internet Computing department Big Data Bites led by Jimmy Lin [20]. Furthermore, this department, which is known to '*deliver thought-provoking and potentially controversial ideas about all aspects of big data*', mentions that Marz's so-called 'invention' was already proposed by Butler Lampson's paper in 1983 [21].

Lambda's architecture follows the pipeline illustrated in Figure 2-2. The first component to be mentioned is the incoming data for processing, designated as Unified Log. It is a set of three generic existing data formats (structured, unstructured, and semi-structured). The idea behind this component is to group all the data and apply some transformations [17], such as conversion to a specific data format for easier communication for the speed layer, the second component. As soon as data come in, they are conducted to two different layers, the Batch layer and the Speed layer, respectively. The speed layer is responsible of processing the data, whereas the batch layer does not process the data at all, maintaining it immutable. Furthermore, the batch layer indexes the batch view to the serving layer and then, the speed layer just updates the already existing serving layer with the most recently processed data. The end-user (either a human or an analytics engine) can visualize the processed data through a web interface like a dashboard.

Digital companies like Yahoo and Netflix [22] use this implementation for their business models. Clearly these types of companies make use of machine learning algorithms in order to obtain the most suitable and updated movies/series preferences list in app or streaming services.

2.2.3 Kappa Architecture

As soon as the Lambda architecture gained its form across the Big Data world, Jay Kreps (2014), a staff engineer at LinkedIn [23], wrote an article where he pointed out some pitfalls to the Lambda architecture. According to Jay, Kappa is a simplification of Lambda and not a replacement [14]. The main difference between these two architectures consists in removing the batch layer and, therefore, automatically decreasing complexity (Figure 2-3). This is achieved avoiding two separate codebases, one for the batch component and the other for the speed component. This means that only one layer (speed layer) is needed and no synchronization between batch and speed is necessary. Everything goes through a single stream (i.e., the speed layer in Figure 2-3).

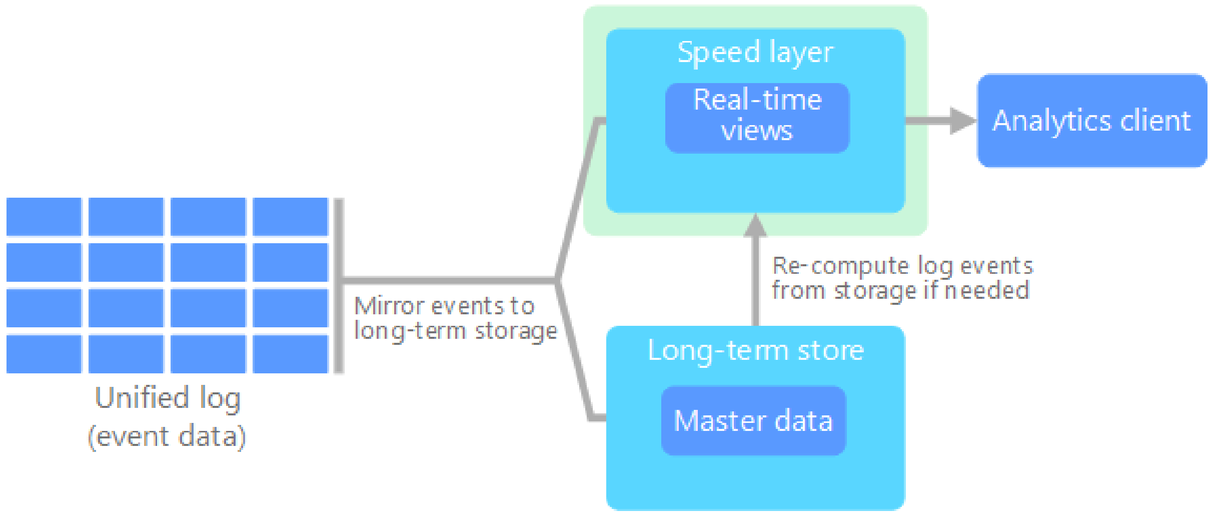


Figure 2-3: Kappa Architecture [14]

In what concerns development frameworks and tools, both Lambda and Kappa can be produced with Apache Kafka and Apache Spark technologies [9] [10]. Along with these two frameworks, there are other possible solutions meaning that there is a large number of technologies enabling to develop these systems [9].

Regarding κ pipeline, it is very similar to λ's architecture. The collected data, represented as the Unified Log in Figure 2-3, passes through the speed layer where the data is processed by the stream process. With Apache Kafka and Apache Spark, it is possible to get the processed data and deliver it and index it to the real-time views, which is similar to the Serving Layer in

Lambda. From there on, analytics clients or final human users can access these data submitting appropriate queries.

2.2.4 Differences and Similarities

λ and κ are in fact two possible solutions to develop a powerful platform for data analytics and Big Data. Indeed, through a deep research and mainly focusing on recent papers, it is totally up to the companies to decide which one to choose and, therefore, which one is the most suitable to their business type [10]. Nevertheless, we should not forget about their specific strengths and weaknesses as summarized in Table 2-1.

Table 2-1: λ and κ Characteristics

Architecture	Strengths	Weaknesses
Λ	More accuracy results due to batch processing	High latency by batching
	Machine Learning effectiveness	Difficult to implement and to keep data and the two codebases synchronized
	Re-computation of data in failure cases	Two codebases (for batch and stream)
	Parallelism	Lack of availability in the batch layer [10]
K	Easier implementation	Evidence of errors during data processing (redundancy, loss data even '*fault-tolerant*')
	Computationally Cheaper	Evidence of errors while updating database
	Single stream for processing (no need for two codebases)	Difficult to migrate and adapt to λ (if it is the case)

To summarize, both architectures are two possible solutions for a data analytic platform. However, no matter what business type it is, it should be planned carefully based on some criteria. One of the criterium that should be considered is that Lambda delivers more accurate processed data but suffers from high latency when batching all the existing data. Kappa, on the other hand, is much easier to implement since it does not require to have two distinct codebases because it does not have the batch layer. It also makes keeping data synchronized easier since there is just a single stream to worry about.

2.2.5 Business Type

There have not been any recent studies or any studies at all to clarify if there is any ideal architecture to any specific business type. Throughout the literature, there are quite a lot of business types coming to foreground with ideas to develop such a data analytics platform. The most quoted business types found were banking and credit companies, e-commerce companies, marketing and streaming services, Mobile Network Operators, and Healthcare. One common factor to all these companies is that they want to contribute with good decisions and strategies. This said, every business type can perfectly consider any of these architectures. They just have to go over their characteristics, which are summarized in Table 2-1, and decide which one to use. If for instance, the importance to maintain data registered and apply any type of machine learning algorithm to get better and accurate results then Lambda is more precise to do so. Otherwise, if the data is not so important, then Kappa is more appropriate to use.

The next section describes two case studies for two distinct business types.

2.2.6 Case Studies

As already mentioned in the previous section, two different case studies will now be described focusing on Netflix, a well-known streaming service, and on the telecommunications industry (Telco Industry). Both follow a different architecture with Netflix using λ as its base architecture [24], among other components, and the Telco Industry preferentially using κ [25].

2.2.6.1 Netflix

Being now one of the most requested streaming services, Netflix already exists since 1997 and has gained popularity along these past years offering several movies and series through streaming. In recent years, streaming has won distinctive appreciation and therefore Netflix has not stopped since then and is now quoted to stream over 97k hours of video every minute [22]. Netflix also keeps track of users' preferences in terms of movies and series, which is achieved through machine learning and personalization algorithms [26], in order to improve their service quality and maximize clients' satisfaction.

The reason why Netflix chose Lambda over Kappa is that the streaming service uses the best of both worlds in just one platform, namely batch and stream processing. It takes advantage of ML algorithms since data is much more precise due to have batching, and by this, it meets Netflix's quality of service standards [27].

2.2.6.2 Telco Industry

In telecommunication industries, where data is constantly present, it is deeply appreciated having a big data platform that can collect data and turn it into useful information. In this case study detecting anomalies is precious to avoid high maintenance costs.

So, Telco companies [25], probably not all, but just reflecting on this source, use κ as an alternative architecture because the batch layer results they want can be obtained using a streaming engine, for instance, Apache Kafka, which is responsible of ingesting volumes of data and therefore ensures the same results as the stream layer [25], classifying batch processing as a subset of stream processing [25].

Since in this case the batch and speed layers would contain the same codebases, it would be redundant and, therefore, unnecessarily to have both codebases synchronized, avoiding extra work on keeping constantly everything synchronized since batch and speed would give similar results. This way, it decreases the complexity of the architecture and respects the removal of the batch layer (Table 2-1), emphasizing this way the speed layer (Figure 2-3). Besides this reason, the article also mentions possible ML algorithms implementations.

3 Requirement analysis and architecture design

This chapter not only is intended to present the architectural design of the real-time Data Analytic platform developed during the internship but also its functional and non-functional requirements. To accomplish all the necessary functionalities of such a platform it is important to specify all the requirements, which represent the functionalities of the pipeline must do to stream data in real-time (section 3.4) and quality attributes of the pipeline (section 3.5). All the requirements were accordingly agreed with the CSW's supervisor to focus on the main goal of the system.

3.1 Architecture

Based on the two analyzed Big Data architectures, the project we developed follows the Kappa architecture which by nature is appropriate for real-time data streaming. However, and considering its main architecture as described in section 2.2.3 and Figure 2-3, it is necessary to tailor it to be able to represent the pipeline for this concrete project. Despite Figure 2-3 does not include literally the serving layer, the 'Analytics client' can be considered as the serving layer since the data is grouped and displayed on the web interface. The tailoring part reflects on that assumption and there is no need to build extra components. The components are built upon existing tools and technologies that can be visualized in Chapter 4.

Figure 3-1 presents the main components that were developed and are explained in section 3.2 (colored in red). It follows the standard structure of the Kappa architecture (colored in grey) as already defined in section 2.2.3.

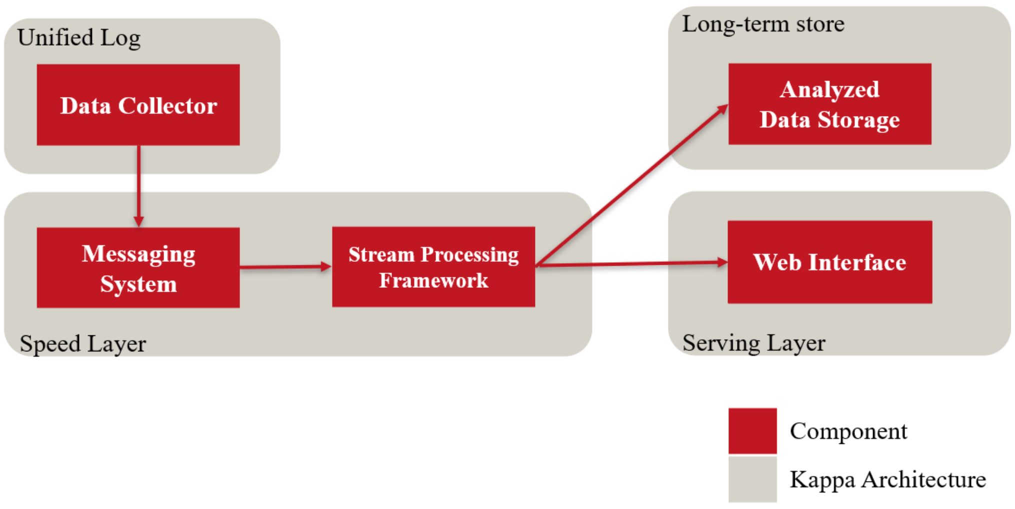


Figure 3-1: Real-Time Data Analytic High-Level Pipeline

3.2 Architecture Components

Once the architecture is defined, this section presents the components' structure, their goal of having it in the pipeline, and their own achievements. For that, follows a list covering all the five essential components.

Data Collector component: This component is the starting point of the system[5], being responsible to generate randomly data based on an existing dataset and according to the given use case that is further explained in section 3.3 and to be consumed by the Messaging System component.

Messaging System component: This component encapsulates the typically known producer and consumer of a messaging system. In other words, its main goal is to be prepared to handle the generated data from Data Collector and, as a producer, send the data to the consumer, which is the Stream Processing Framework. For this component, Apache Kafka is used to accomplish the process of producing and consuming the data.

Stream Processing Framework component: This component is our so-said consumer, and therefore our manager for receiving the data from the messaging system. Once the data has arrived, the data is consumed and pulled to a non-mandatory process, which is known for executing transformations[6] (part of the Extract-Transformation-Load, i.e., ETL, process) on the incoming data. For this component, Apache Spark is the core technology that is able to produce ETL processes and send them through existing libraries to the final two components: Analyzed Data Storage and Web Interface components, respectively.

Analyzed Data Storage component: This component is more than just a regular data storage or data repository. Its responsibility is to store and save all the incoming data in their RAW[7] format and the processed data, if passed through the ETL process, in databases according to the project's needs.

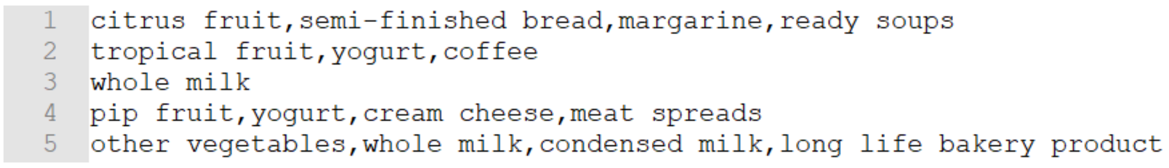
Web Interface component: To conclude the project's pipeline, it should be possible to visualize the processed data. Thus, the web interface component is very important since it is the Visualization/Serving layer of the entire project and the crucial window in which conclusions are drawn for business.

3.3 Use case

Building a data pipeline system as described in the previous section without any concrete or precise data to support the analytical part of the project would not be conclusive nor would result in lessons to be learned. Therefore, Pedro Miranda, the CSW supervisor for this project, defined a specific use case that is based on a grocery e-commerce. The idea was to represent a

[5] In this context it is the pipeline
[6] Transformations on data such as aggregations, cleansing and filtering
[7] original

demo of a grocery's transactions e-commerce shop that enables to graphically present which country has made more transactions in that website, which product has been bought the most and how many transactions were made, among other interesting graphs including the purchased items of each transaction. The names of the items were supported by an existing dataset [28] and its content can be partially visualized in Figure 3-2, which is just a small subset of the entire dataset.



```
citrus fruit,semi-finished bread,margarine,ready soups
tropical fruit,yogurt,coffee
whole milk
pip fruit,yogurt,cream cheese,meat spreads
other vegetables,whole milk,condensed milk,long life bakery product
```

Figure 3-2: Subset of the grocery dataset

According to the dataset, each line represents a transaction with x products. There are no other attributes in the dataset but the names of the purchased items. Since this project's use case is about groceries, the idea behind this dataset is the extraction of the product names to be stored afterwards into a regular database, or relational database technically spoken.

The dataset contains n transactions and obviously any product would be part of m transactions, meaning that the dataset would contain replicas of product names. Removing duplicates must be handled before storing the names of the products into the database so therefore, it suffered some transformation, such as cleansing. By underdoing this step, it is possible to speak of an ETL process. This achievement is explained in a more detailed way in section 4.6.

3.4 Functional requirements

This use case contains some basic and generic functionalities that were already mentioned and understood while establishing the literature review (Chapter 2), such as having a way to transmit data from one location to another, transform data or store it in databases. Table 3-1 reports all the developed functional requirements and are grouped by component approach.

While for the Data Collector, it is important to generate randomly data based on an existing dataset, the Messaging System component is able to read those generated data and publish them so that the Stream Processing Framework can read the data and apply some transformations. To display graphs with the data, it is important to send the processed data from Stream Processing Framework to the Analyzed Data Storage component and also to the Web Interface component. Furthermore, other functional requirements were needed to help generate the random data as well as a Data Mining algorithm is applied on the processed data that is stored in the database.

3.5 Non-Functional requirements

Like any other software project, this project also includes some non-functional requirements (NFR), which basically represent the quality attributes of the system. In this case, quality attributes refer to Availability, Capacity, Security, Usability, Scalability, or Reliability just to name a few.

Within this project, consistency, efficiency, and scalability are main NFR. As already specified by the name of the project, a real-time data analytics platform must handle data in fast speed. Therefore, it must be efficient. Scalability is important when dealing with big data. There must be always a huge structure to support big data volumes. Consistency is crucial because the idea is not just to send data from one location to another, but also to do something with that data. In that case, ETL processes can help by removing redundancy from the data and, thus, facilitate the application of ML algorithms or other detection pattern algorithms.

The approach to prove the efficiency of the NFR within this project's scope is based on the chosen tools and technologies used for each of the Kappa architecture components. More details regarding each quality attributes of the components are specified in Chapter 4. Any tool that is used within this project does have quality attributes that satisfies the project's quality and ensures the good functioning of such a real-time data analytic platform.

Table 3-1: Functional requirements

#	Functional requirement
1 Data Collector	
1.1	Generate random grocery e-commerce Data
2 Messaging System	
2.1	Retrieve Data using Apache Kafka
2.2	Send the incoming data to the Messaging System
3 Stream Processing Framework	
3.1	Consume Data from the Messaging System
3.2	Apply transformations to the data
3.3	Send processed Data to the Analyzed Data Storage component
3.4	Send Data to the Visualization layer (part of Web Interface component)
4 Analyzed Data Storage	
4.1	Create Databases to store processed and raw Data
5 Web Interface	
5.1	Create visualization plots
6 Data Mining	
6.1	Build REST API in Python
6.2	*Apriori* Algorithm application to stored Data
7 ETL Dataset	
7.1	Apply ETL process within existing dataset

4 Implementation

This chapter intends to present individual aspects of each component. Not only does each section indicate how they communicate with each other, but also explains what each other's input and output results are. Along with all reasonable justifications, Figure 4-1 helps to perceive its general flow and indicates the main technologies, tools or frameworks used.

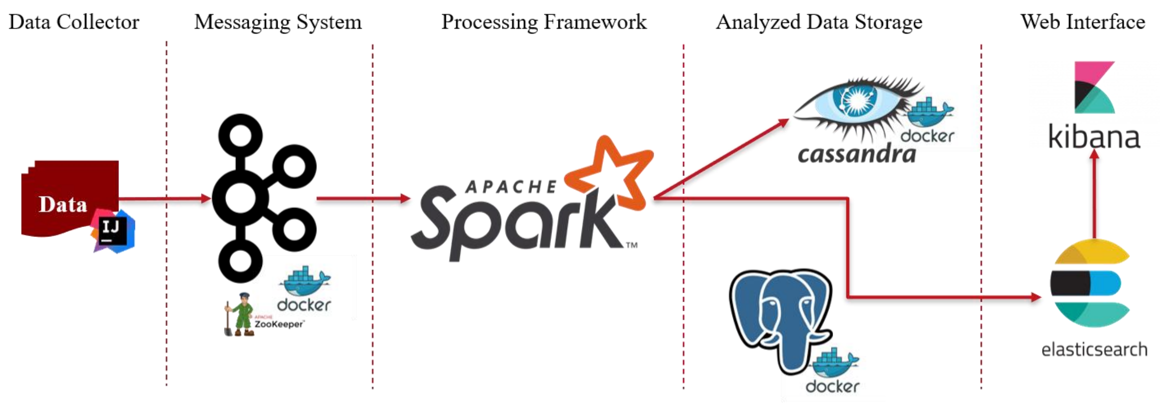


Figure 4-1: Pipeline Projection

Regarding the tools and frameworks used for each individual component, a more detailed description can be encountered in section 4.1 under Table 4-1. Also, initial setup and configuration needs are described in section 4.2 to understand base requirements crucial to run this pipeline. All other relevant technical aspects can be found in Appendix C.

4.1 Tools and Technologies

As previously mentioned, this section includes the tools and the technologies used for the grocery e-commerce use case. In this context, these tools were chosen while reviewing the literature and suggested by the CSW supervisor Álvaro Menezes. For a better reading experience, Table 4-1 is divided into several chunks to differentiate the importance of each layer and to group them by their main goal. Their versions as well as their use within the project are referenced by the 'Version' and 'Use' columns, respectively. It is important to mention that this use case depends on numerous technologies not previously known by the student. This fact represented the main challenge and a great investment in research was required.

Table 4-1: Tools, Frameworks and Technologies used within the project

Tools and Frameworks			
Name	Version	Description	Use (*cf 3.4*)
Apache Kafka	2.5.0	Producer and Consumer mechanism, to publish Data and to be subscribed by Apache Spark respectively	Messaging System
Apache Zookeeper		Management of resilient distributed data offering availability and scalability	
Apache Spark	2.4.6	Consumer that reads the data and processes them by applying ETL processes	Processing Framework
Docker/ Docker Desktop	2.2.0.0	*Docker-compose* was used to deploy and run instances of services that are needed by the project	Messaging System, Analyzed Data Storage
Flask	1.1.2	Web service gateway Interface to host methods concerning the *Apriori* algorithm (see last row in this table)	Data Mining
Elasticsearch	7.7.1	Elasticsearch is used to index the data so that Kibana can collect the data and do Visualization interface	Web Interface
Kibana			
IDE and Programming Languages			
Name	Version	Description	Use
IntelliJ - Java	2019.3.2 (Community Edition)	Java programming language used for implementing Data Collector, Messaging System and Stream Processing Framework	Data Collector, Messaging System, Processing Framework
PyCharm - Python	2020.1.1 (Community Edition)	Python programming language used for implementing REST API using the *Apriori* algorithm and the ETL Pipeline for initial dataset	Data Mining, ETL Dataset
Repository Storage and Version Control			
Name	Version	Description	Use
Bitbucket	7.4	Version Control System for the entire code	Entire Project

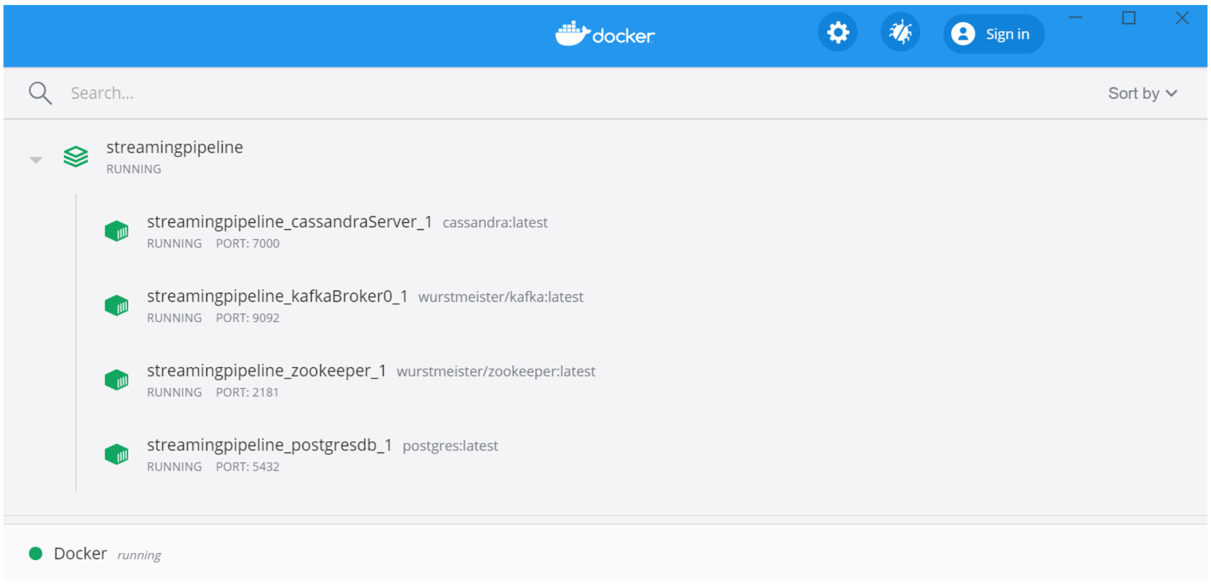
Cassandra	2.5.0	Store Big Data (NoSQL); data that is processed through the pipeline is stored in Cassandra	Analyzed Data Storage
PostgreSQL	42.2.14	Store the groceries existing dataset after ETL apply (*cf* 3.4 ETL Dataset and 4.6)	Analyzed Data Storage
Jira	8.13	Product backlog containing FR and project evolution	Entire Project
Data Mining			
Name	Version	Description	Use
Apriori	1.1.1 [29]	Data Mining algorithm that has been used to detect pattern on the groceries e-commerce	Data Mining

4.2 Setup

Inevitably, it was necessary to download and to try out some tools and frameworks mentioned in Table 4-1 since at the beginning there was no experience from the past.

Docker is a so-called base layer for some pipeline components, namely Messaging System and Analyzed Data Storage. In other words, docker serves as a virtual machine to deploy the pipeline. Inside that virtual instance, four different docker containers were created based on existing images[8]. For the Messaging System, two instances were needed, namely an Apache Kafka Broker[9], which is identified inside a Cluster[10], and Apache Zookeeper which comes automatically with Apache Kafka since its responsibility is to control and manage the broker(s) inside the cluster. It then can allocate and/or re-allocate the broker within the cluster, assuming a better performance regarding scalability and availability.

A *docker-compose* file was created and is provided in this report as an appendix (Appendix B). That file contains instructions so that all docker containers are run simultaneously[11], avoiding the need for any other additional configuration or run them one by one. Figure 4-2 represents a snapshot of the Docker Desktop application with all four docker containers running.



streamingpipeline
RUNNING

streamingpipeline_cassandraServer_1 cassandra:latest
RUNNING PORT: 7000

streamingpipeline_kafkaBroker0_1 wurstmeister/kafka:latest
RUNNING PORT: 9092

streamingpipeline_zookeeper_1 wurstmeister/zookeeper:latest
RUNNING PORT: 2181

streamingpipeline_postgresdb_1 postgres:latest
RUNNING PORT: 5432

● Docker *running*

Figure 4-2: Docker Desktop with running containers

Regarding the *docker-compose* file, the instructions are built based on tags. These tags are necessary to specify the services to be created, which are the containers. For example, the Kafka broker is built on an existing image called *wurstmeister/kafka* and it depends on the Zookeeper

[8] Technical name given in a docker context which represents e.g., an instance of a tool
[9] Like a single instance of a client
[10] Group of brokers (it can also be a single node/machine)
[11] Command to run under command line is: *docker-compose up*

service, which is also built based on the *wurstmeister/zookeeper* image. These images can be downloaded and looked up on the official Docker hub repository [30].

The broker follows some specification, such as environment variables, so that it is possible to create topics[12], send the data, which is created locally on the machine and sent over to the created topic, and be ready to be read by the Apache Kafka technology (producer/consumer mechanism). More information covering this topic is provided in section 4.4, which is about the Messaging System component.

For the data to flow from the Data Collector through the Messaging System, a new topic is needed. This is achievable by specifying the *KAFKA_CREATE_TOPICS* tag, which in this case is given the name *topic1*. It is also to mention that these services are run inside a specific Internet Protocol (IP) address after creating a network (*kafkaNetwork*) with the subnet address 172.19.0.0/16. To mention that, this subnet address was chosen to be the docker network address since it was one of the available addresses to run all docker services. These specifications allow the communication between containers and maintaining the structure of the Messaging System organized.

In this specific use case, only one broker is necessary, but of course this differs from project to project when it comes to complexity and requirements. Another interesting and relevant element that is found on the *docker-compose* file is the definition of two distinct databases. More details concerning it are described in section 4.6.

4.3 Data Collector

Data Collector, also denominated Producer (Figure 4-7), is based on the Java object-oriented programming language, and, since there was some previous experience with, the chosen Integrated Development Environment was IntelliJ (Figure 4-3).

Figure 4-3: Data Collector Component

To simplify this use case, two Plain Old Java Object (POJO) classes were built in order to help generate all necessary data for this component. They are identified by Figure 4-4 and Figure 4-5. A POJO class permits to group all necessary attributes in one Java class, making it accessible to any Java program to use the POJO class ensuring the reuse of it in any other project and for instance store the attribute values in a data storage (in this context, the POJO class is stored in the database).

[12] Similar to a pipe, gets input and returns output

All POJO classes implement the *Serializable* interface, which permits to transform Java classes into series of bytes and send them over to the Messaging System under the Apache Kafka technology. *ProductUDT* is an auxiliary POJO class containing only two distinct attributes that are detailed in Table 4-2.

```
public class ProductUDT implements Serializable {
    private Integer idproduct;
    private String productname;
…
```

Figure 4-4: Java class ProductUDT

Table 4-2: ProductUDT POJO class

Class: ProductUDT	
Attribute	Use and Definition
idproduct	Product identifier (unique values)
productname	Product name (unique names)

The *Order* class is a more complete and representative state of a transaction of the grocery e-commerce (Figure 4-5). It has multiple attributes that are specified in greater detail in Table 4-3. It is important to mention that this class contains a list of *ProductUDT* (see previous POJO class). This class is the transaction/order that goes to the Stream Processing Framework since transformations are applied to this specific *Order* POJO class (cf. section 4.5)

```
public class Order implements Serializable {
    private UUID uuid;
    private Integer idOrder;
    private String items;
    private Date timestamp;
    private List<ProductUDT> products;
    private String countryOfOrigin;
…
```

Figure 4-5: Java class Order

Table 4-3: Order POJO class

Class: Order	
Attribute	Use and Definition
Uuid	Universally Unique Identifier (*UUID)*
idOrder	Order identifier
Items	String containing items by their id product (e.g. 1,3,6,18,56,…)
Timestamp	Date when the Order has been purchased
Products	Product information coming from the database
countryOfOrigin	Reflects to the country where the Order has been purchased

4.4 Messaging System

For the Messaging System component, Apache Kafka has been used as the main technology (Figure 4-6). Kafka's approach is based on a producer/consumer architecture as mentioned in Figure 4-7 and its ecosystem works under the docker name *streamingpipeline_kafkaBroker0_1* (Figure 4-2).

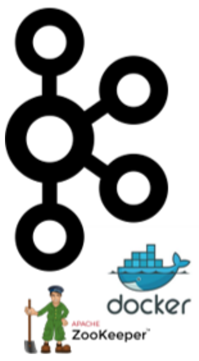


Figure 4-6: Messaging System component

Beside Apache Kafka, Apache Zookeeper also contributes to this component by controlling scalability and availability, running under the docker name *streamingpipeline_zookeeper_1* (Figure 4-2).

Figure 4-7 represents the architecture the Messaging System offers. On the left side, the previous component, Data Collector publishes (or sends) the data to the Kafka broker, which forwards it to the created topic. As soon as the data comes in, the other service, Zookeeper container, provides the required scalability and availability so that the data is not run over. The broker subsequently waits for the consumer to subscribe to its topic so that it can read the data. In this case, Apache Spark is responsible for consuming the data and proceeding to the transformations upon the data, called ETL process.

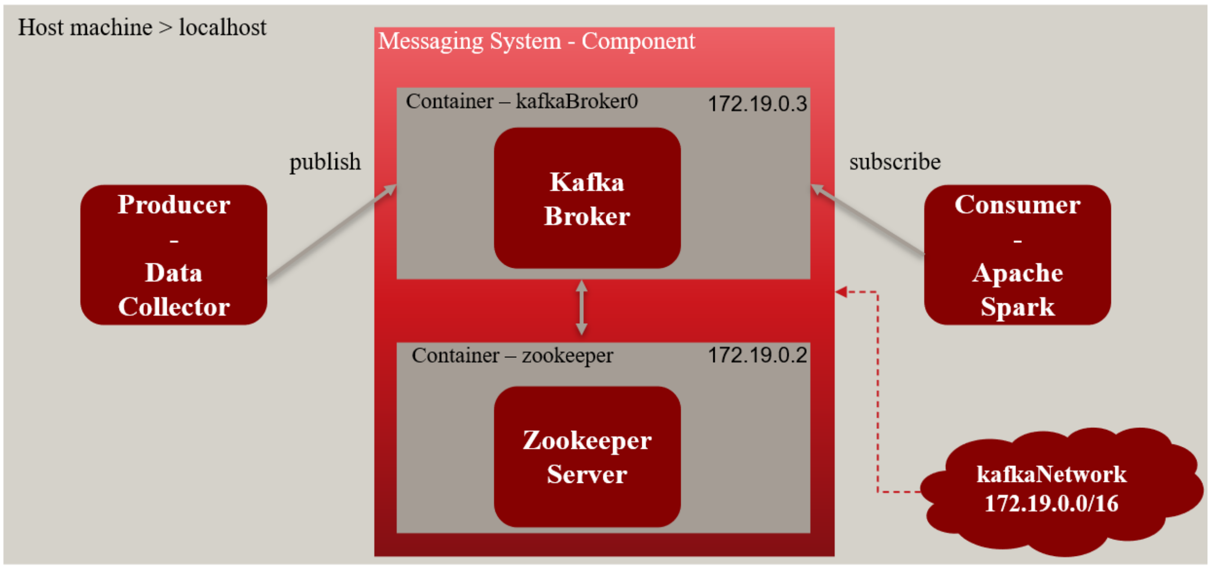


Figure 4-7: Producer/Consumer within docker containers

4.5 Stream Processing Framework

In the market, there is a variety of existing stream processing frameworks, such as Apache Flink, Apache Samza, Apache Storm, and Apache Spark just to name a few [31]. For this specific project, Apache Spark was the selected framework. Despite all the other mentioned frameworks could integrate and work for this project or similar projects, Apache Spark was more complete by offering more libraries and modules, which resulted in its selection. The content of Table 4-4 [31] , which provides a comparison between processing frameworks based on six features, also supports the choice of Apache Spark.

Table 4-4: Stream Processing Framework Comparison

	Characteristics					
Stream Processing Framework	Delivery Guarantees	Fault tolerance	State Management	Performance	Advanced Features	Maturity
Apache Flink	✓	✓	✓	✓	✓	⊖
Apache Samza	✕	✓	✓	✓	✕	✓
Apache Storm	✕	✓	✕	⊖	✕	✓
Apache Spark	✓	✓	✓	✓	✓	✓

Legend:

✓ Strong | ✕ Weak | ⊖ So-so

Based on Table 4-4, Apache Flink and Apache Spark have both the best ranks guaranteeing fault-tolerance and good performances. By this, it is stated that these frameworks can retake their operation after failure (energy, network, node). They are also the best solutions for getting higher throughput and low latency. The chosen framework was in this context preferred by having a much wider community than Apache Flink, which also confirmed by the first assigned CSW supervisor.

Apache Spark was the selected framework and therefore, it must be well understood before proceeding to the implementation of it. The component is responsible to receive inputs and deliver outputs, the generated data coming from the Messaging System and the stored procedure to the Data Storage and Web Interface respectively.

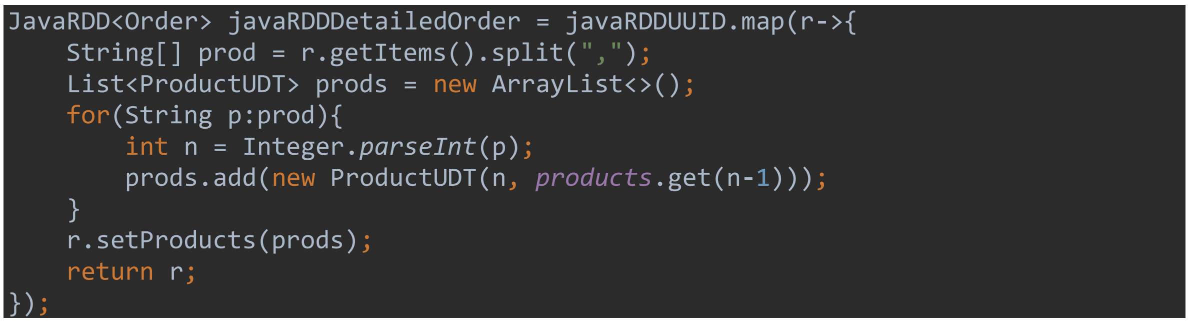
Concerning the inputs, Apache Spark has a module called Spark Streaming (Table 4-4) which is the core system of this component. Its job is to be aware of incoming data, endlessly. This means that this specific component is connected (subscribed) to the Kafka broker, through the existing Kafka Integration API within Spark Streaming, and receives an alert, which technically is a `ConsumerRecord` object. These objects are key-value tuples where the identification and the generated data from the Data Collector are stored. Whenever there is an alert, a Resilient Distributed Dataset (RDD) is created. RDDs are known to be immutable data structures in Spark. Their life cycle aims to maintain the data as it came and represent a distributed collection of objects. In this case, these objects inherit from the user-defined class *Order* (Figure 4-5). Whatever the change is, small or huge, on that RDD, a new RDD is always created. This means that the very first RDD has been copied to a brand new RDD and that both the old and new RDD are accessible whenever evoked.

For instance, the example in Figure 4-8 demonstrates that each RDD that comes through the `directKafkaStream` object is mapped into a collection of type *Order*, making it the first RDD. After deciding to apply a UUID to that distributed collection, a second RDD is created, *javaRDD* and *javaRDDUUID* respectively.

```
directKafkaStream.foreachRDD(rdd -> {
    System.out.println("--- New RDD with " + rdd.partitions().size() + "
partitions and " + rdd.count() + " records");
    JavaRDD<Order> javaRDD = rdd.map(ConsumerRecord::value);
    JavaRDD<Order> javaRDDUUID = javaRDD.map(r->{
        r.setUuid(UUID.randomUUID());
        return r;
    });
```

Figure 4-8: [Code] Apache Spark code with ETL process

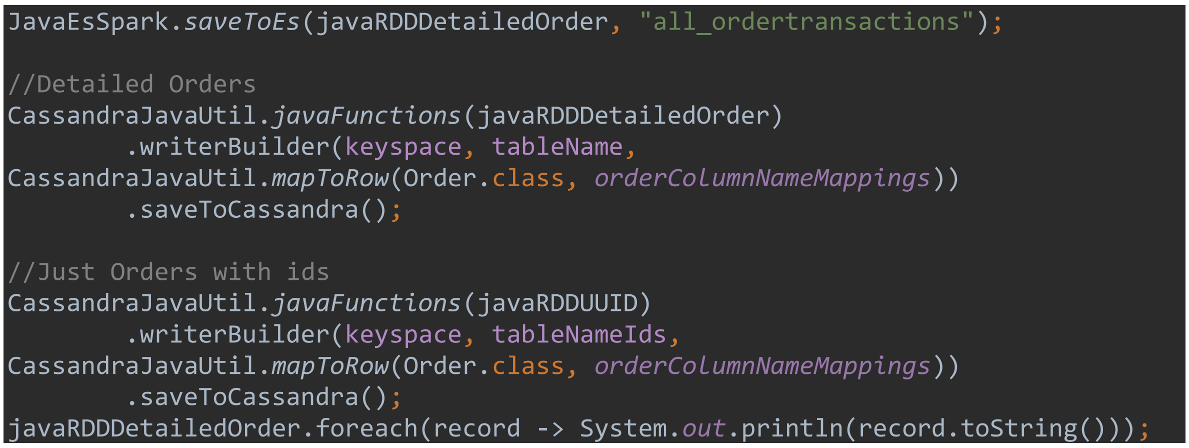
Now that every RDD has a distinct UUID, the biggest transformation is yet to come. Coming back to the Data Collector component, every incoming Order has a field called *items*. Its content is of type *String*, which means that this attribute contains a string composed by product identification numbers (that are stored in the postgresql database, *cf* Section 4.6). In order to know which identifier corresponds to the product name in the postgres database, the code section in Figure 4-9 splits every product identifier to a String array *prod*, where finally every product is added to an empty dynamic array of *ProductUDT*. This means that *javaRDDDetailedOrder* is filled with the corresponding product name from the database.



```
JavaRDD<Order> javaRDDDetailedOrder = javaRDDUUID.map(r->{
    String[] prod = r.getItems().split(",");
    List<ProductUDT> prods = new ArrayList<>();
    for(String p:prod){
        int n = Integer.parseInt(p);
        prods.add(new ProductUDT(n, products.get(n-1)));
    }
    r.setProducts(prods);
    return r;
});
```

Figure 4-9: [Code] Detailed Order on Apache Spark

To store every processed and final RDD (*javaRDDDetailedOrder*), two actions are performed, namely indexing them to the Elasticsearch tool and saving them in two different tables in the Cassandra database: the detailed order table and the only-id order (*javaRDDUUID*) table (Figure 4-10).



```
JavaEsSpark.saveToEs(javaRDDDetailedOrder, "all_ordertransactions");

//Detailed Orders
CassandraJavaUtil.javaFunctions(javaRDDDetailedOrder)
        .writerBuilder(keyspace, tableName,
CassandraJavaUtil.mapToRow(Order.class, orderColumnNameMappings))
        .saveToCassandra();

//Just Orders with ids
CassandraJavaUtil.javaFunctions(javaRDDUUID)
        .writerBuilder(keyspace, tableNameIds,
CassandraJavaUtil.mapToRow(Order.class, orderColumnNameMappings))
        .saveToCassandra();
javaRDDDetailedOrder.foreach(record -> System.out.println(record.toString()));
```

Figure 4-10: [Code] Storing data from Apache Spark to Databases

When accessing localhost:4040 through a web browser (e.g., Chrome), Apache Spark offers a graphical DAG (Directed Acyclic Graph) visualization (Figure 4-11) containing all the ETL processes, including the creation of the Kafka DirectStream object that consumes the incoming data, as well as the map transformations. In other words, the first blue rectangle in Figure 4-11 is the creation of the `directKafkaStream` object (Figure 4-8) and the other three map transformations are done by running the code shown in Figure 4-8 and Figure 4-9. Once the transformations come to an end, the RDD containing the POJO class (*Order*) is stored to the Cassandra database and also to the Elasticsearch service (Figure 4-10).

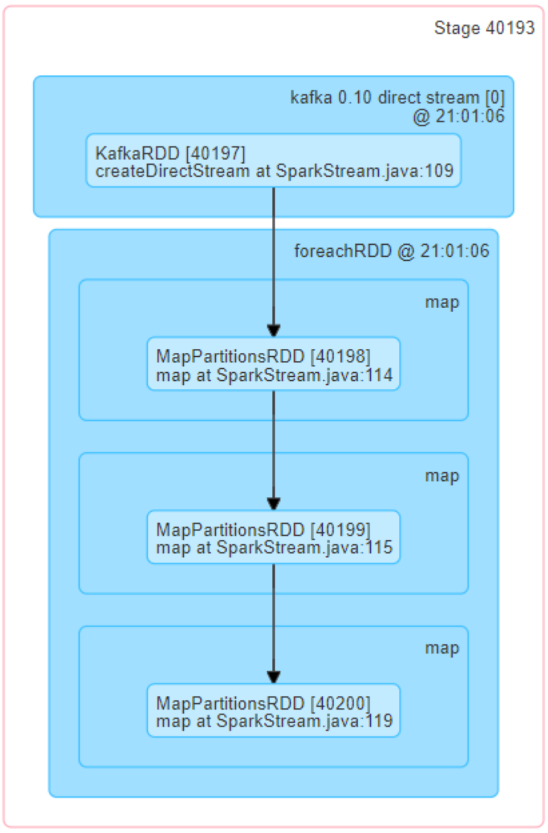

Stage 40193
kafka 0.10 direct stream [0] @ 21:01:06
KafkaRDD [40197] createDirectStream at SparkStream.java:109
foreachRDD @ 21:01:06
map
MapPartitionsRDD [40198] map at SparkStream.java:114
map
MapPartitionsRDD [40199] map at SparkStream.java:115
map
MapPartitionsRDD [40200] map at SparkStream.java:119

Figure 4-11: Apache Spark DAG Visualization

This concludes the process of the stream processing framework component by having read the incoming data, conducted an ETL process and stored the processed data to its final storage location.

4.6 Analyzed Data Storage

In a very generic perspective, this component stores all the data involved in the project. To separate the data which has been generated randomly based on the dataset from what is static throughout the project, both data types follow different paths, emphasizing Big Data and ‘regular’ Data, respectively. This distinction is required since the amount of generated data is expected to be significantly higher than the amount of regular data that can be supported by a regular database. For the Big Data, the considered database is the NoSQL [13] Cassandra management system and for the regular database a PostgreSQL Database is considered. Using Apache Cassandra as a database (Figure 4-12) does not require the data to be normalized and therefore no relationships are needed between Cassandra tables. Of course, these advantages are not the only ones nor the single reason it was chosen for the project. As stated in the official website of Apache Cassandra, scalability, high availability and fault-tolerance are ensured [32].

[13] Non-relational database

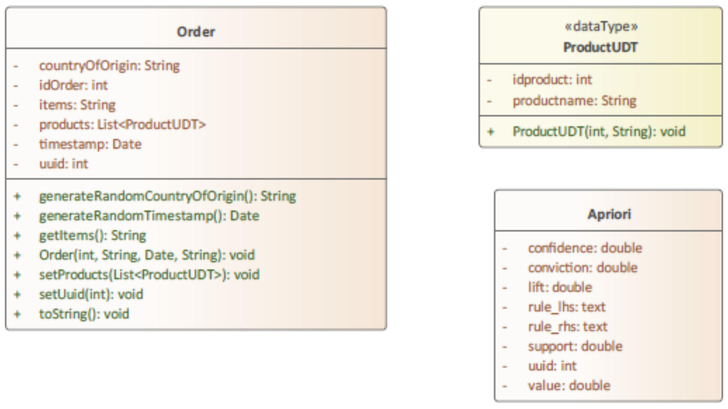


Figure 4-12: Cassandra database

Regarding the relational database, the existing e-commerce products were retrieved from an existing data source (*csv* file format) which was illustrated in Figure 3-2. Since each row represents a transaction, there might be redundancy along the rows when it comes to the products. Hence, a python script was developed to eliminate any duplicate from file and only retain distinct products. Through this approach, it is possible to identify a pipeline, or ETL pipeline to be more precise. The script ingests the data (extract the data from the *csv* file), applies the needed transformation, which is removing any duplicate, and, finally, the output is sent over to the relational database.

Products (Figure 4-13) is the main and only entity in the postgres database. It retains the *productName*, which is guaranteed by the ETL pipeline explained previously, and contains other basic information like a product description (*productDescription*) and a quantity for that product. Finally, all products are identified with a unique identifier (*idProduct*).

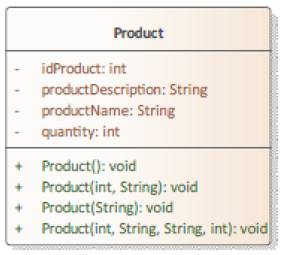


Figure 4-13: PostgreSQL database

4.7 Web Interface

This component is visually the most appealing and the most revealing part of this project. It does not only present the insights of what has been ordered and purchased, but also contributes to the Data Analyst's study field whose goal is to take the most out of these facts and numbers.

The aim of this component is to help Data Analyst's to extract insights of any company by applying Data Analytic knowledge upon interesting and promising graphs such as bar, column, or pie charts. ELK stack, Elasticsearch, Logstash and Kibana contribute to this component by indexing incoming data, processing, and designing attractive dashboards over the user interface (UI).

In fact, just Elasticsearch and Kibana (EK) are used in this project. Logstash did not appear to be revealing since its main generic objective is to simulate a pipeline that can ingest, transform, and output the processed data. This, of course, would not make any difference since our goal is to create a pipeline that can do all of this by using more powerful and scalable technologies, such as Apache Kafka and Apache Spark which promise a better, stable, scalable job [33].

Regarding Elasticsearch and Kibana, both tools were supported and run on the local machine, which also could have been run on official existing docker images [34] [35]. Once they are running, it is possible to access them through a web browser under `localhost:5601`. The Kibana app pops up and presents its interface with the graphs dashboard.

One important aspect was to understand how Elasticsearch and Kibana work together. At first, the data that is sent to Elasticsearch is indexed and is stored based on documents. In other words, the index value represents the repository where n documents are saved, which can be considered as single rows in the perspective of a regular database. This means every document is represented by the attributes that were generated by the Data Collector and, when sent to Elasticsearch, the mapping of data types is done automatically.

Regarding Kibana, it offers many interactive charts that can be assigned appropriately to the data to be presented. In this case, Table 4-5 contains all the questions that can be used to build graphs on the Kibana tool.

Table 4-5: Data Analytics Business Questions

Questions
What is the most bought product?
Which country has purchased more?
How many transactions were made in total?
At what time are made more transactions?
Average transactions per country?
How many transactions per hour?

4.8 Data Mining

Data Mining is not directly related with in any of the components described in the previous sections. It is a side activity in the context of the project described in this report. Even though Apache Spark offers multiple Machine Learning (ML) libraries with a variety of algorithms, it was decided by the CSW supervisor to use one of the earliest existing Data Mining algorithms, the *Apriori* algorithm [36].

Globally, Data Mining technics are applied to an existing dataset or to a data warehouse to detect any pattern. Within the grocery e-commerce case study, it focuses in analyzing the Cassandra database and detect possible patterns. Since this is all about e-commerce products, it totally makes sense to apply this algorithm and understand how each product can relate with other products. For instance, when purchasing a specific product, which other products can grab a user's attention and possibly make part of the final shopping list just before checkout? The idea is to find that pattern and then, help the 'company' to get some advantages, such as business opportunities. For instance, applying a discount on a product (chips) when knowing that if another product (beer) is bought, then the first product has more probability to be bought (confidence) with the second product, meaning that this could lead to a positive profit.

This algorithm is set with 4 heuristics, namely: Support, Confidence, Lift, and Conviction. Each one contributes to a final prediction and certain confidence about a product that has been purchased and/or is related to. The results of this algorithm are stored in the Apriori table inside the Cassandra database (Figure 4-12). Table 4-6 describes the four heuristic.

Table 4-6: Heuristics of the Apriori algorithm [37]

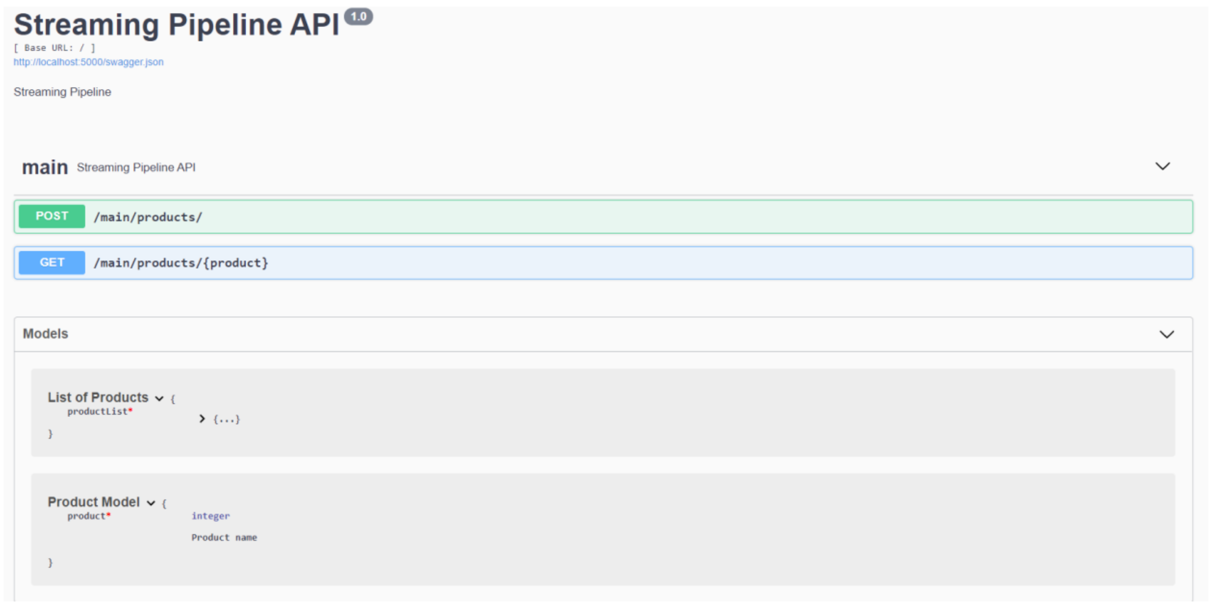
Heuristic	Interval	Description	Formula
Support	[0, 1]	Number of times that product X occurs in transactions	$S\{X\}$
Confidence	[0, 1]	Probability that Y is purchased given X was purchased	$C\{X \rightarrow Y\} = \dfrac{S\{X \rightarrow Y\}}{S\{X\}}$
Lift	[0, +∞]	Likelihood of Y being purchased when X is purchased, while considering the popularity of Y	$lift\{X \rightarrow Y\} = \dfrac{S\{X \rightarrow Y\}}{(S\{X\} * S\{Y\})}$
Conviction	[0, +∞]	Ratio between X and Y, being both independent	$conv\{X \rightarrow Y\} = \dfrac{(1 - S\{X\})}{(1 - C\{X \rightarrow Y\})}$

4.9 REST API

A REST API (Representational State Transfer Application Programming Interface) is an architectural style for an API to communicate under http requests with other applications [38]. The retrieved data over the requests can be achieved by using commands such as GET, POST, PUT and DELETE. For this specific use case, the API we developed has a single URI (Uniform Resource Identifier) that is related to section 4.8 (Data Mining) and implements the *Apriori* algorithm. The purpose of this feature complements this use case by stating and detecting frequent item set patterns and therefore giving insight about the business.

This feature is built using the Python programming language within the PyCharm IDE. For the creation of the web server gateway interface, Flask framework has been used. The API is prepared to trigger off an action when receiving a POST http request. The requests can be a list of one or more products and the body of the response, in JSON (JavaScript Object Notation) format, contains a complete list with other products that are related to the sent list based on the *Apriori* algorithm. For instance, using the web e-commerce example, this would give the customer other related products that could also be purchased just before checkout.

The prototype of the POST method follows as an example in Figure 4-14 and is ready to trigger off when receiving such a http request. The product list, as previously mentioned, is identified as *productList* and consists of one or more product ids (type: integer). For instance, in case of having the product id 25 (e.g., beer) in the *productList* list and if the method is called, the result which can be obtained are products that have quite good relationship with product number 25. Maybe, hungry programmers might consider chips as a good deal and reconsider buying it just before checkout.


Streaming Pipeline API 1.0
[Base URL: /]
http://localhost:5000/swagger.json

Streaming Pipeline

main Streaming Pipeline API

POST /main/products/

GET /main/products/{product}

Models

```
List of Products {
    productList*        > {...}
}

Product Model {
    product*            integer
                        Product name
}
```

Figure 4-14: REST API for the *Apriori* algorithm

5 Results and Test Cases

During the time allocated to the internship no unit tests or integration tests for the components were performed due to time limitation. Therefore, this chapter only presents some results regarding the developed Kibana graphs as well as a demonstration on how the *Apriori* algorithm behaves exposing some test cases. In this phase, the real-time data analytic platform is totally built by all necessary components and is therefore ready to be executed. The Data Collector starts to generate random data based on the existing dataset, that are further analyzed and transformed to build detailed orders using the specified POJO class *Order* (Figure 4-5). This said, the graphs contain these generated data, and these are displayed in the Kibana dashboard. Kibana provides a range of graphs and these are created by drag and drop. This setup does not require any programming skill, just a correct configuration is needed, since the data from Elasticsearch must be indexed correctly to the created dashboard.

5.1 Kibana Graphs

Kibana tool is responsible to automatically update the dashboard and all the graphs that are inside the dashboard. While the Data Collector generates random data repeatedly, like an endless circle, other components do their work, inclusive updating the data on the Elasticsearch so that Kibana can update the data on the graphs automatically. These plots are represented by different figures, starting from Figure 5-1 to Figure 5-5, and each figure does represent a question that can be found in Table 4-5.

Figure 5-1 presents an overview counter with the number of transactions.

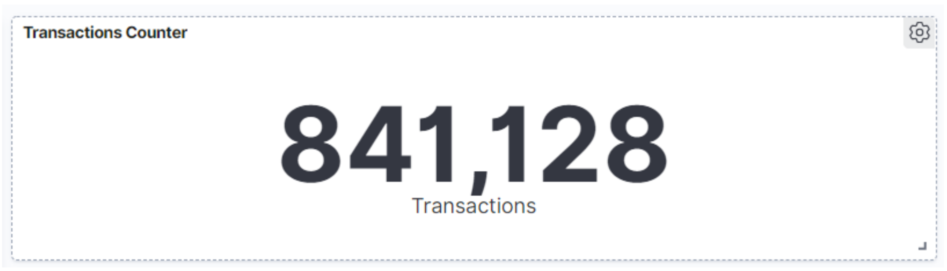


Figure 5-1: Number of transactions counter

Figure 5-2 is a donut chart in which the countries regarding the transactions are displayed. A country that does not figure in the top 10 transactions is stored in the ‘Other’ category. When hovering over a specific country on the graph, it is possible to get more details, such as the exact number of transactions per country.

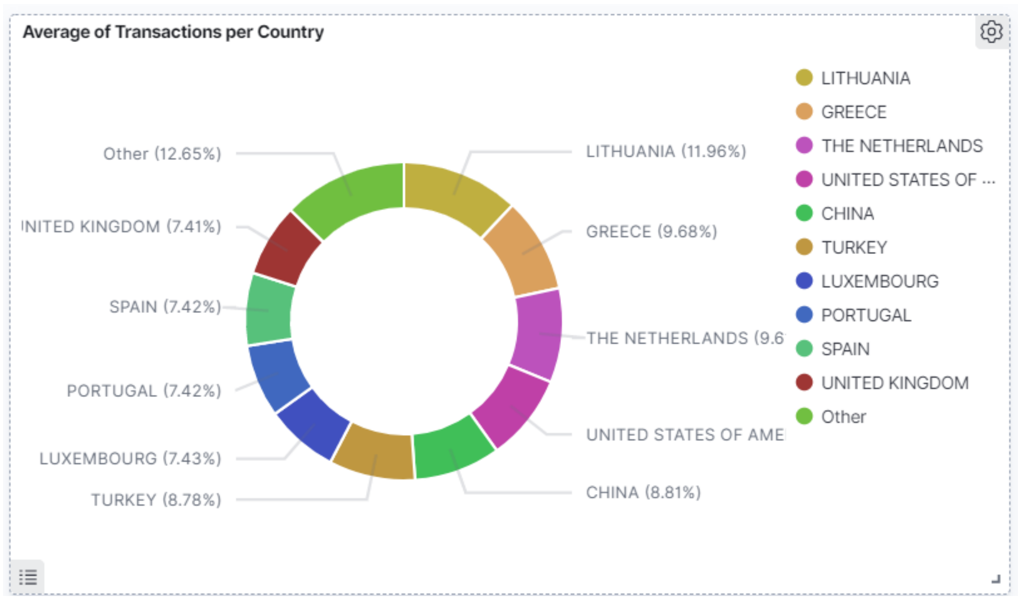


Figure 5-2: Average of Transactions per Country donut chart

Figure 5-3 is tabular chart where the representation follows the same path as the previous figure (Figure 5-2).

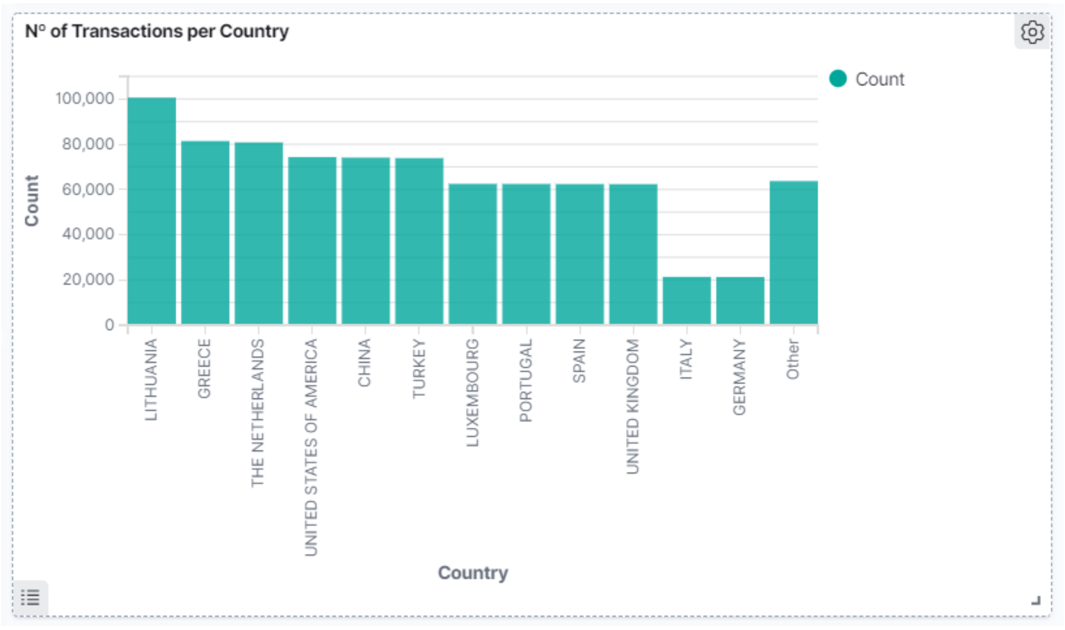


Figure 5-3: Number of transactions per country plot

Figure 5-4 displays the 10 most bought products regarding all the transactions.

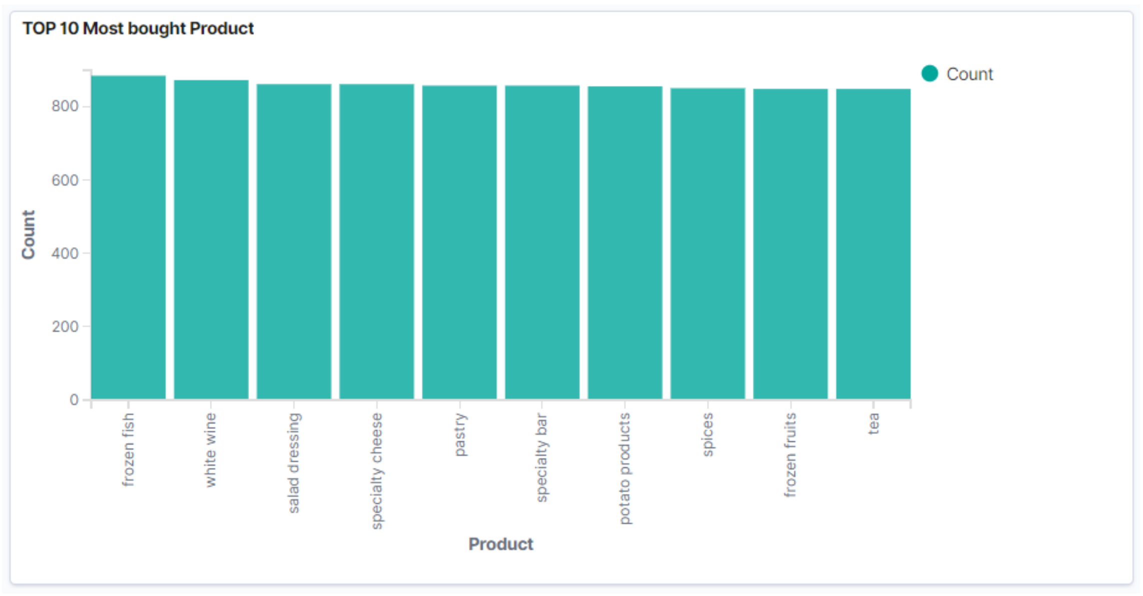


Figure 5-4: Top 10 most bought product plot

Figure 5-5 is a line plot that intends to give a feedback covering the transactions on a per hour base. This time range is about a day long and is divided into periods of exactly one hour.

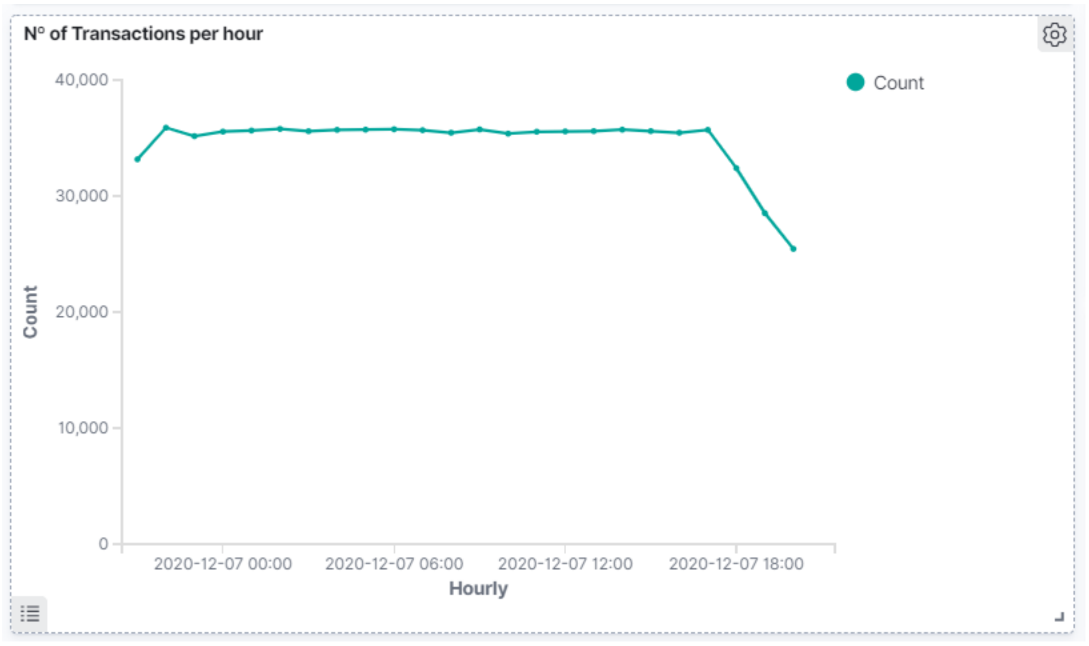
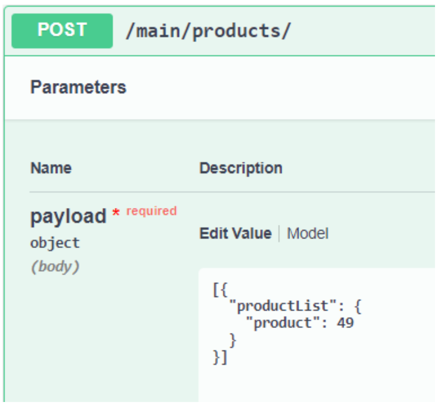


Figure 5-5: Number of transactions per hour line plot

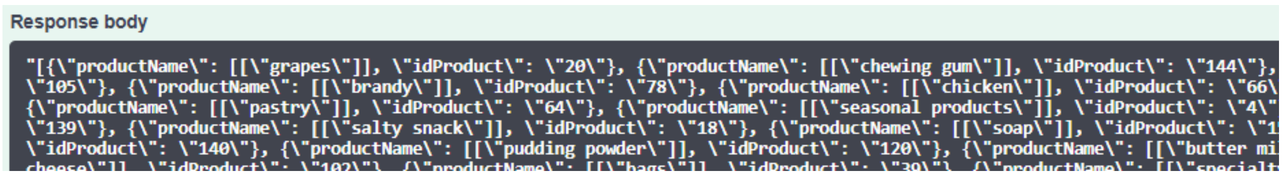
5.2 *Apriori* test results

As already mentioned, the *Apriori* algorithm was implemented as a REST API (Figure 4-14). Figure 5-6 presents an example of the single URI of the developed REST API being accessed throw a POST http method. It specifies the purchased product number 49 in the URI parameter (Frozen fish, which is the most bought item [Figure 5-4]). The body of the http response message, in JSON format, is presented in Figure 5-7 and contains the names and respective identifiers of the products that figure the most when buying product number 49. It retrieves the item with the highest interest/relevance first. In this case, grapes, chewing gum and fruit/vegetable juice are the three most requested items when buying frozen fish, according to the *Apriori* algorithm. Despite this being an example within this project's use case, a pattern is here identified and is a Data Analyst's job to get more insights to the company by producing more of these results.



```
[{
  "productList": {
    "product": 49
  }
}]
```

Figure 5-6: REST API call



```
"[{\"productName\": [[\"grapes\"]], \"idProduct\": \"20\"}, {\"productName\": [[\"chewing gum\"]], \"idProduct\": \"144\"},
\"105\"}, {\"productName\": [[\"brandy\"]], \"idProduct\": \"78\"}, {\"productName\": [[\"chicken\"]], \"idProduct\": \"66\
{\"productName\": [[\"pastry\"]], \"idProduct\": \"64\"}, {\"productName\": [[\"seasonal products\"]], \"idProduct\": \"4\"
\"139\"}, {\"productName\": [[\"salty snack\"]], \"idProduct\": \"18\"}, {\"productName\": [[\"soap\"]], \"idProduct\": \"1
\"idProduct\": \"140\"}, {\"productName\": [[\"pudding powder\"]], \"idProduct\": \"120\"}, {\"productName\": [[\"butter mi
cheese\"]], \"idProduct\": \"102\"}, {\"productName\": [[\"bags\"]], \"idProduct\": \"39\"}, {\"productName\": [[\"specialt
```

Figure 5-7: Result of the POST verb on the URI /main/products

6 Conclusions and Future Research directions

This section aims at drawing some conclusions about the work described in this document and discusses directions for future research.

6.1 Conclusions

The constant technological evolution and the increasing amount of Data on the web tend to bring changes to existing real-time data analytic platforms. Data science is therefore in constant change. With the aim to continue delivering good performance levels and/or to maintain the already existing quality of service of the companies using these platforms, the project described in this report intended to discuss about two major existing Big Data architectures, Lambda and Kappa, and perceive if they are generic[14] enough solutions suitable to support real-time data analytic platforms.

The literature review, the first task performed, allowed to dive deeper to the existing problem and to study topics that could be helpful to achieve the purpose of the research goals. It also established the state of the art and comparison of Big Data solutions that effectively support real-time data analytics platforms, namely Lambda and Kappa. Then, two real business cases were presented to illustrate the use of Lambda and Kappa in a streaming service (Netflix) and in the telecommunication industry, respectively, and draw some conclusions. Finally, an implementation of a real-time data analytic platform based on the Kappa architecture has been achieved by using a grocery e-commerce use case which was further explored using the *Apriori* algorithm to discover frequent item sets pattern.

The main theoretical contribution of this work is a detailed description, analysis, and comparison of two Big Data architectures, Lambda and Kappa, for real-time data analytics. The features, strengths, and weaknesses of both were clearly explained, as well as real business cases presented.

The practical contributions consisted in following the Kappa architecture to implement a straightforward processing pipeline, with all the essential components implemented, to support data analytics for an e-commerce grocery use case. A large number of tools and frameworks, such as Apache Kafka and Apache Spark, were used to this end. Most of them were new to the trainee. With this practical hands-on, the Kappa architecture was put on proof to understand if the concepts and key principles studied by the theoretical contribution were correctly applied. This includes the use of all components proposed by its generic architecture and the use of the studied technologies that figure in published articles.

The main challenges of this project are summarized as follows. Its scope, i.e., Data Engineering, was not within the student's study field and was different from any previous project at Critical

[14] Suitable to be applied to any business type

Software, the host institution. Therefore, the student had to learn and work with never-seen technologies, frameworks, tools, and programming languages, as well as relevant Big Data and other theoretical concepts. The global pandemic also affected the project itself. The first allocated CSW supervisor, Álvaro Menezes, had to be replaced mid second semester, which resulted in schedule adjustments. The software development methodology was also changed and working from home was not as effective as from CSW's offices. Also, a difficult familiar situation led to a delay on the project, specifically on the documentation writing phase. Nevertheless, the goal of this project was successfully achieved and CSW has now one base framework for further developments. Consequently, CSW accomplished their goal that was to have this pipeline structure as generic as possible that ensures its implementation to any business type.

6.2 Directions for future research

To conclude this chapter, this section presents some possible approaches and directions for future developments and research. Therefore, the following list introduces some of these features.

During the development phase, Apache Kafka offered a variety of APIs, including processing methods (*Kafka DStreams*). Back then, a stable version was not confirmed and thus made the decision fall on Apache Spark (Table 4-4) for the Stream Processing Framework component. Meanwhile, Apache Kafka could have extended its Kafka DStreams library making it possible to use Kafka as for the Messaging System component as well as for the Stream Processing Framework component. This would make Apache Spark's redundant and not worth anymore to be implemented despite their enriched libraries (e.g., Machine Learning, Spark Streaming). As a future research it would be interesting to perceive if it is effective to apply Apache Kafka to the entire speed layer, including Stream Processing Framework and Messaging System (removing Apache Spark from the project).

This project focused an e-commerce use case and the *Apriori* algorithm was implemented to detect a pattern on the products. However, using the machine learning library from Apache Spark and applying any other probability distribution (i.e., not uniform) on the input data might also be a reasonable approach to detect patterns. This approach might make this project even more realistic.

Another future research direction can be identified in the serving layer. The way the data is displayed and visualized through the Kibana tool is not the best offer in the market. Instead, another possible visualization tool could be used, such as Tableau [39], Power BI [40], or others, replacing Elasticsearch and Kibana. Moreover, Kibana presents some difficulties on displaying the indexing documents on the plots as well as some limitations by not offering more customizations, such as add labels on the plots and grouping the documents, change color just to name a few.

References

[1] "Instituto Superior de Engenharia de Coimbra," [Online]. Available: https://www.isec.pt/PT/Default.aspx. [Accessed 13 July 2020].

[2] "Critical Software," [Online]. Available: https://www.criticalsoftware.com/. [Accessed 13 July 2020].

[3] "Culture CSW," 2020. [Online]. Available: https://www.criticalsoftware.com/en/our-world/culture-history.

[4] "CSW History," 2020. [Online]. Available: https://www.criticalsoftware.com/en.

[5] "ISEC - The Institute," [Online]. Available: https://www.isec.pt/EN/ISEC/.

[6] ISEC, "Fikalab - ISEC," [Online]. Available: https://fikalab.pt/isec/. [Accessed 16 November 2020].

[7] Z. Milosevic, W. Chen, A. Berry and F. A. Rabhi, "Real-Time Analytics," p. 1, December 2016.

[8] S. Melendez and A. Pasternack, "Here are the data brokers quietly buying and selling your personal information," February 2019. [Online]. Available: https://www.fastcompany.com/90310803/here-are-the-data-brokers-quietly-buying-and-selling-your-personal-information.

[9] K. N. Singh, R. K. Behera and J. K. Mantri, "Big Data Ecosystem - Review on Architectural Evolution," ResearchGate, 2019.

[10] M. Feick, N. Kleer and M. Kohn, "Fundamentals of Real-Time Data Processing Architectures Lambda and Kappa," pp. 3, 6, 2018.

[11] S. Bandyopadhyay and A. Banik, "Big Data - A Review on Analysing 3Vs," *Journal of Scientific and Engineering Research,* pp. 3-4, 2016.

[12] S. Sheriff, "UNDERSTANDING THE 5VS OF BIG DATA," March 2019. [Online]. Available: https://acuvate.com/blog/understanding-the-5vs-of-big-data/.

[13] C. Tozzi, "Big Data 101: Dummy's Guide to Batch vs. Streaming Data," July 2017. [Online]. Available: https://blog.syncsort.com/2017/07/big-data/big-data-101-batch-stream-processing/.

[14] Microsoft, "Big data architectures," Microsoft, December 2018. [Online]. Available: https://docs.microsoft.com/en-us/azure/architecture/data-guide/big-data/.

[15] A. Awad, S. Sakr and E. Shahverdi, "Big Stream Processing Systems: An Experimental Evaluation," *ResearchGate,* p. 54, April 2019.

[16] A. J. Awan, M. Brorsson, V. Vlassov and E. Ayguade, "Architectural Impact on Performance of In-memory Data Analytics: Apache Spark Case Study," ResearchGate, 2016.

[17] S. Salloum, R. Dautov, X. Chen, P. X. Peng and J. Z. Huang, "Big data analytics on Apache Spark," Springer International, Switzerland, 2016.

[18] N. Marz, "LinkedIn," [Online]. Available: https://www.linkedin.com/in/nathanmarz.

[19] N. Marz, "How to beat the CAP theorem," *thoughts from the red planet,* 13 October 2011.

[20] J. Lin, "The Lambda and the Kappa," *Big Data Bites*, pp. 60-66, October 2017.

[21] B. W. Lampson, "Hints for Computer System Design," Xerox Palo Alto Research Center, 1983.

[22] N. Hunt, "Netflix & AWS Lambda Case Study," in *https://aws.amazon.com/solutions/case-studies/netflix-and-aws-lambda/,* 2014.

[23] "Jay Kreps at Linkedin," [Online]. Available: https://www.linkedin.com/in/jaykreps. [Accessed 14 July 2020].

[24] D. Bryant, "Migrating Batch ETL to Stream Processing: A Netflix Case Study with Kafka and Flink," February 2018. [Online]. Available: https://www.infoq.com/articles/netflix-migrating-stream-processing/.

[25] N. Seyvet and I. M. Viela, "Applying the Kappa architecture in the telco industry," 19 May 2016. [Online]. Available: https://www.oreilly.com/ideas/applying-the-kappa-architecture-in-the-telco-industry.

[26] D. Gray, "Netflix's Big Data Architecture," 15 May 2014. [Online]. Available: https://dataconomy.com/2014/05/netflix-big-data-architecture/.

[27] I. Samizadeh, "A brief introduction to two data processing architectures — Lambda and Kappa for Big Data," 15 March 2018. [Online]. Available: https://towardsdatascience.com/a-brief-introduction-to-two-data-processing-architectures-lambda-and-kappa-for-big-data-4f35c28005bb.

[28] N. Caballero, "Github," [Online]. Available: https://github.com/stedy/Machine-Learning-with-R-datasets/blob/master/groceries.csv. [Accessed 13 April 2020].

[29] A. algorithm, "pypi," [Online]. Available: https://pypi.org/project/efficient-apriori/. [Accessed 14 June 2020].

[30] "Docker Hub Repository," [Online]. Available: https://hub.docker.com/. [Accessed March 2020].

[31] C. Prakash, "Spark Streaming vs Flink vs Storm vs Kafka Streams vs Samza : Choose Your Stream Processing Framework," [Online]. Available: https://www.linkedin.com/pulse/spark-streaming-vs-flink-storm-kafka-streams-samza-choose-prakash. [Accessed 2 March 2020].

[32] Apache Cassandra, [Online]. Available: https://cassandra.apache.org/. [Accessed 25 September 2020].

[33] "XPLG," [Online]. Available: https://www.xplg.com/what-is-logstash/. [Accessed 28 September 2020].

[34] "Docker Hub Repository - Elasticsearch," [Online]. Available: https://hub.docker.com/_/elasticsearch. [Accessed November 2020].

[35] "Docker Hub Repository - Kibana," [Online]. Available: https://hub.docker.com/_/kibana. [Accessed November 2020].

[36] Y. Djenouri, "Combining Apriori Heuristic and Bio-Inspired Algorithms for Solving the," *Information Sciences,* p. 3, August 2017.

[37] tommyod, "Efficient-Apriori Documentation," November 2020. [Online]. Available: https://readthedocs.org/projects/efficient-apriori/downloads/pdf/latest/.

[38] M. Rouse, C. Bedell, E. Hannan, S. Wilson and A. Gillis, "RESTful API (REST API)," [Online]. Available: https://searchapparchitecture.techtarget.com/definition/RESTful-API. [Accessed December 2020].

[39] "Tableau Official Website," [Online]. Available: https://www.tableau.com/. [Accessed December 2020].

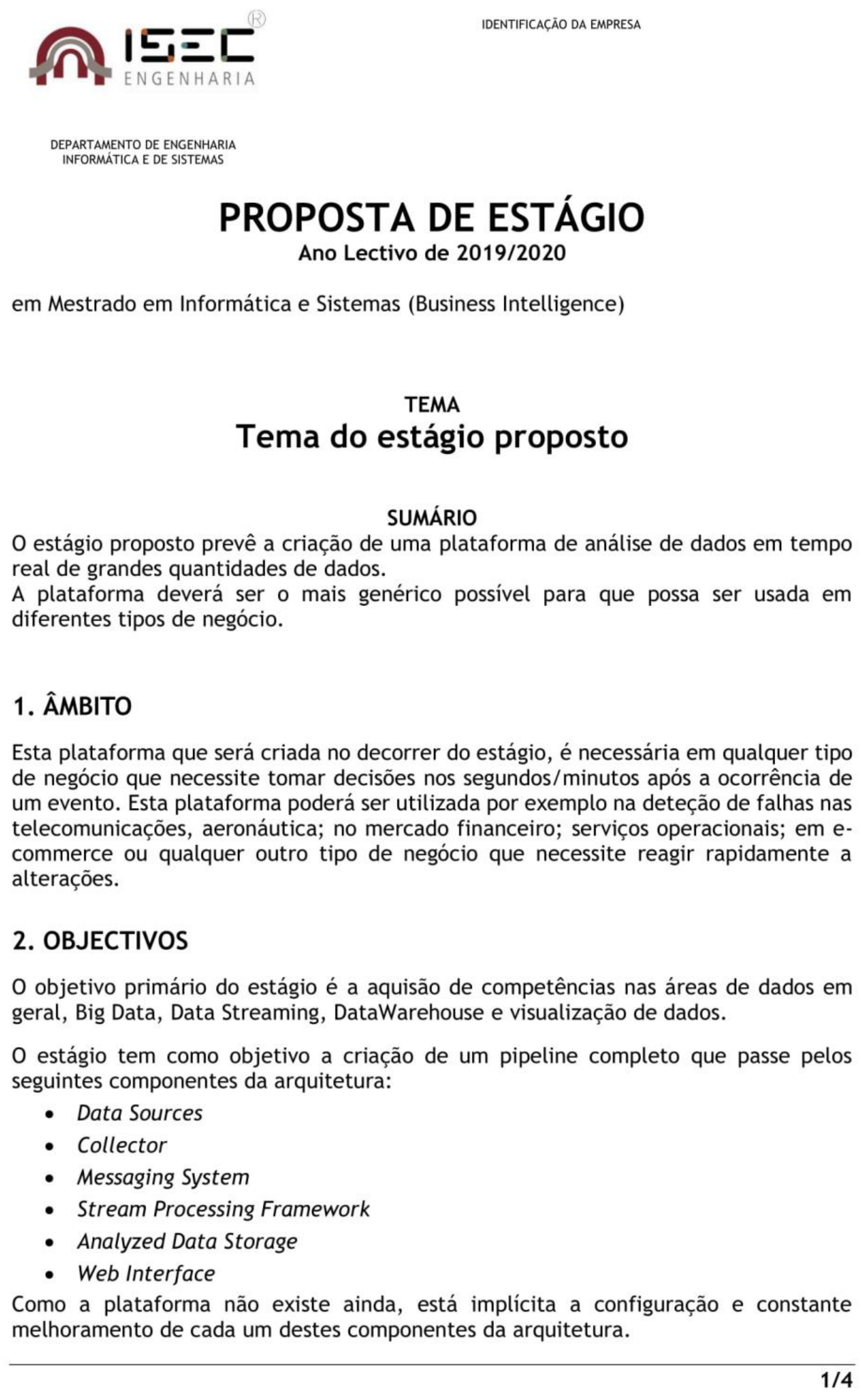
[40] "Power BI Official Website," [Online]. Available: https://powerbi.microsoft.com/en-us/. [Accessed December 2020].

[41] R. Felder and R. Silverman, "Learning and Teaching Styles in Engineering Education," *Journal of Engneering Education 78 (7),* pp. 674-681, 1988.

[42] E. v. Heck and P. Vervest, "Smart business networks: how the network wins," *Communications of the ACM, Vol. 50 No. 6,* pp. 28-37, 2007.

Appendix A – Internship proposal

Appendix A: Internship proposal



IDENTIFICAÇÃO DA EMPRESA

ISEC ENGENHARIA

DEPARTAMENTO DE ENGENHARIA
INFORMÁTICA E DE SISTEMAS

PROPOSTA DE ESTÁGIO

Ano Lectivo de 2019/2020

em Mestrado em Informática e Sistemas (Business Intelligence)

TEMA

Tema do estágio proposto

SUMÁRIO

O estágio proposto prevê a criação de uma plataforma de análise de dados em tempo real de grandes quantidades de dados.
A plataforma deverá ser o mais genérico possível para que possa ser usada em diferentes tipos de negócio.

1. ÂMBITO

Esta plataforma que será criada no decorrer do estágio, é necessária em qualquer tipo de negócio que necessite tomar decisões nos segundos/minutos após a ocorrência de um evento. Esta plataforma poderá ser utilizada por exemplo na deteção de falhas nas telecomunicações, aeronáutica; no mercado financeiro; serviços operacionais; em e-commerce ou qualquer outro tipo de negócio que necessite reagir rapidamente a alterações.

2. OBJECTIVOS

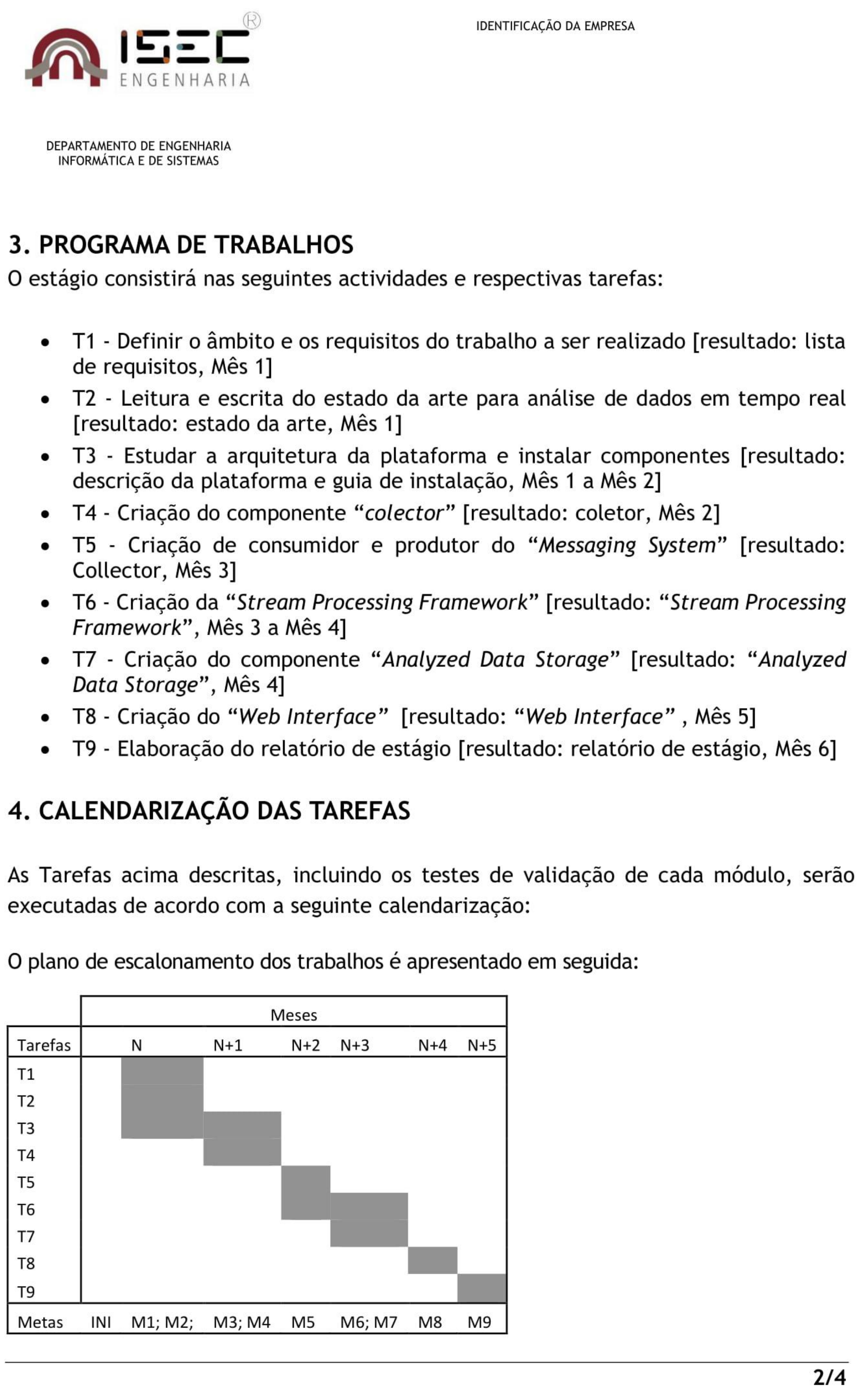
O objetivo primário do estágio é a aquisição de competências nas áreas de dados em geral, Big Data, Data Streaming, DataWarehouse e visualização de dados.

O estágio tem como objetivo a criação de um pipeline completo que passe pelos seguintes componentes da arquitetura:

- *Data Sources*
- *Collector*
- *Messaging System*
- *Stream Processing Framework*
- *Analyzed Data Storage*
- *Web Interface*

Como a plataforma não existe ainda, está implícita a configuração e constante melhoramento de cada um destes componentes da arquitetura.

1/4



ISEC ENGENHARIA

DEPARTAMENTO DE ENGENHARIA INFORMÁTICA E DE SISTEMAS

IDENTIFICAÇÃO DA EMPRESA

3. PROGRAMA DE TRABALHOS

O estágio consistirá nas seguintes actividades e respectivas tarefas:

- T1 - Definir o âmbito e os requisitos do trabalho a ser realizado [resultado: lista de requisitos, Mês 1]
- T2 - Leitura e escrita do estado da arte para análise de dados em tempo real [resultado: estado da arte, Mês 1]
- T3 - Estudar a arquitetura da plataforma e instalar componentes [resultado: descrição da plataforma e guia de instalação, Mês 1 a Mês 2]
- T4 - Criação do componente "*colector*" [resultado: coletor, Mês 2]
- T5 - Criação de consumidor e produtor do "*Messaging System*" [resultado: Collector, Mês 3]
- T6 - Criação da "*Stream Processing Framework*" [resultado: "*Stream Processing Framework*", Mês 3 a Mês 4]
- T7 - Criação do componente "*Analyzed Data Storage*" [resultado: "*Analyzed Data Storage*", Mês 4]
- T8 - Criação do "*Web Interface*" [resultado: "*Web Interface*" , Mês 5]
- T9 - Elaboração do relatório de estágio [resultado: relatório de estágio, Mês 6]

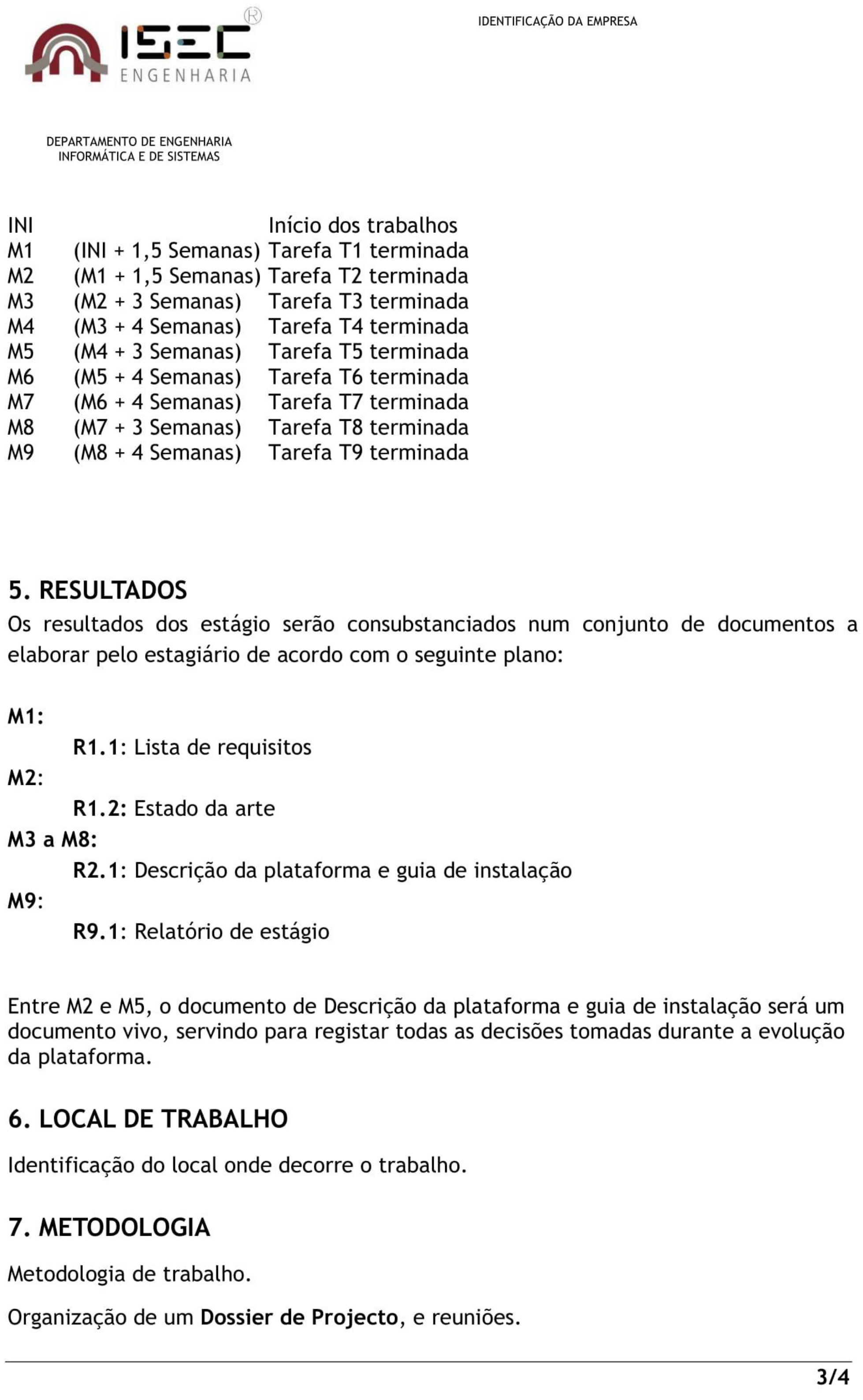
4. CALENDARIZAÇÃO DAS TAREFAS

As Tarefas acima descritas, incluindo os testes de validação de cada módulo, serão executadas de acordo com a seguinte calendarização:

O plano de escalonamento dos trabalhos é apresentado em seguida:

	Meses						
Tarefas		N	N+1	N+2	N+3	N+4	N+5
T1		▨					
T2		▨					
T3		▨	▨				
T4			▨				
T5				▨			
T6				▨	▨		
T7					▨		
T8						▨	
T9							▨
Metas	INI	M1; M2;	M3; M4	M5	M6; M7	M8	M9

2/4



IDENTIFICAÇÃO DA EMPRESA

DEPARTAMENTO DE ENGENHARIA
INFORMÁTICA E DE SISTEMAS

INI		Início dos trabalhos
M1	(INI + 1,5 Semanas)	Tarefa T1 terminada
M2	(M1 + 1,5 Semanas)	Tarefa T2 terminada
M3	(M2 + 3 Semanas)	Tarefa T3 terminada
M4	(M3 + 4 Semanas)	Tarefa T4 terminada
M5	(M4 + 3 Semanas)	Tarefa T5 terminada
M6	(M5 + 4 Semanas)	Tarefa T6 terminada
M7	(M6 + 4 Semanas)	Tarefa T7 terminada
M8	(M7 + 3 Semanas)	Tarefa T8 terminada
M9	(M8 + 4 Semanas)	Tarefa T9 terminada

5. RESULTADOS

Os resultados dos estágio serão consubstanciados num conjunto de documentos a elaborar pelo estagiário de acordo com o seguinte plano:

M1:

R1.1: Lista de requisitos

M2:

R1.2: Estado da arte

M3 a M8:

R2.1: Descrição da plataforma e guia de instalação

M9:

R9.1: Relatório de estágio

Entre M2 e M5, o documento de Descrição da plataforma e guia de instalação será um documento vivo, servindo para registar todas as decisões tomadas durante a evolução da plataforma.

6. LOCAL DE TRABALHO

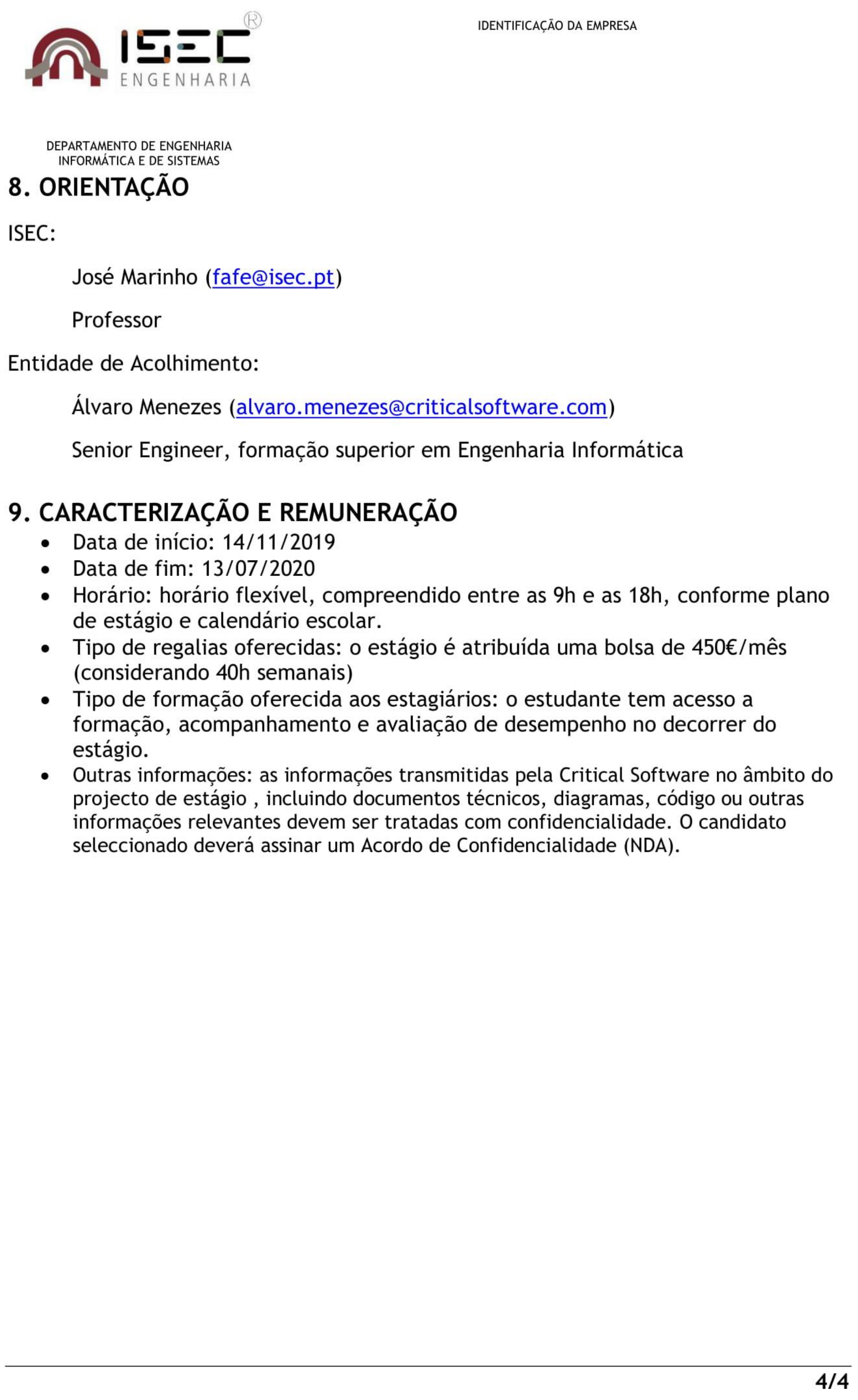
Identificação do local onde decorre o trabalho.

7. METODOLOGIA

Metodologia de trabalho.

Organização de um **Dossier de Projecto**, e reuniões.

3/4



IDENTIFICAÇÃO DA EMPRESA

DEPARTAMENTO DE ENGENHARIA
INFORMÁTICA E DE SISTEMAS

8. ORIENTAÇÃO

ISEC:

José Marinho (fafe@isec.pt)

Professor

Entidade de Acolhimento:

Álvaro Menezes (alvaro.menezes@criticalsoftware.com)

Senior Engineer, formação superior em Engenharia Informática

9. CARACTERIZAÇÃO E REMUNERAÇÃO

- Data de início: 14/11/2019
- Data de fim: 13/07/2020
- Horário: horário flexível, compreendido entre as 9h e as 18h, conforme plano de estágio e calendário escolar.
- Tipo de regalias oferecidas: o estágio é atribuída uma bolsa de 450€/mês (considerando 40h semanais)
- Tipo de formação oferecida aos estagiários: o estudante tem acesso a formação, acompanhamento e avaliação de desempenho no decorrer do estágio.
- Outras informações: as informações transmitidas pela Critical Software no âmbito do projecto de estágio , incluindo documentos técnicos, diagramas, código ou outras informações relevantes devem ser tratadas com confidencialidade. O candidato seleccionado deverá assinar um Acordo de Confidencialidade (NDA).

4/4

Appendix B – Docker-compose file

Appendix B: Docker-compose file

```
version: '3.7'
networks:
  default:
    name: kafkaNetwork
    driver: bridge
    ipam:
      config:
        - subnet: 172.19.0.0/16
services:
  zookeeper:
    image: wurstmeister/zookeeper:latest
    ports:
      - "2181:2181"
    environment:
      ZOOKEEPER_SERVER_ID: 0
      ZOOKEEPER_CLIENT_PORT: 2181
  kafkaBroker0:
    build: .
    image: wurstmeister/kafka:latest
    hostname: kafkaBroker0
    depends_on:
      - zookeeper
    ports:
      - "9092:9092"
    expose:
      - "9093"
    environment:
      KAFKA_BROKER_ID: 0
      KAFKA_ADVERTISED_HOST_NAME: 172.19.0.3
      KAFKA_LISTENERS: LISTENER_DOCKER_INTERNAL://kafkaBroker0:9093,
LISTENER_DOCKER_EXTERNAL://0.0.0.0:9092
      KAFKA_ADVERTISED_LISTENERS: LISTENER_DOCKER_INTERNAL://kafkaBroker0:9093,
LISTENER_DOCKER_EXTERNAL://localhost:9092
      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP:
LISTENER_DOCKER_INTERNAL:PLAINTEXT,LISTENER_DOCKER_EXTERNAL:PLAINTEXT
      KAFKA_INTER_BROKER_LISTENER_NAME: LISTENER_DOCKER_INTERNAL
      KAFKA_CREATE_TOPICS: "topic1:4:1"
      KAFKA_ZOOKEEPER_CONNECT: "zookeeper:2181"
      KAFKA_LOG_DIRS: /kafka/kafka-logs-kafkaBroker0
      KAFKA_DELETE_TOPIC_ENABLE: "true"
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
  cassandraServer:
    image: cassandra:latest
    ports:
      - 7199:7199
      - 7000:7000
      - 7001:7001
      - 9042:9042
      - 9160:9160
    expose:
      - "9042"
    environment:
      - "MAX_HEAP_SIZE=256M"
      - "HEAP_NEWSIZE=128M"
    volumes:
      - cassandra:/var/lib/cassandra
  postgresdb:
    build: .
    image: postgres:latest
    environment:
      POSTGRES_USER: "*****"
      POSTGRES_PASSWORD: "*****"
      POSTGRES_DB: "*****"
    volumes:
      - postgresdb_data:/var/lib/postgresql/data
    ports:
      - 5432:5432
    networks:
      - default
volumes:
  postgresdb_data:
  cassandra:
```

Appendix C – Technical Report

Appendix C: Technical Report

Model Report

Version ●

Date/Time Generated:	06/12/2020 17:16:16
Author:	df-carvalho

EA Repository : C:\Users\df-carvalho\Desktop\RelatorioFinal\diagramas\diagramas.eapx

Table of Contents

Model

Package in package ''

Model
Version Phase 1.0 Proposed
df-carvalho created on 06/12/2020. Last modified 06/12/2020

Class Diagram

Package in package 'Model'

Class Diagram
Version 1.0 Phase 1.0 Proposed
df-carvalho created on 04/12/2020. Last modified 05/12/2020

Cassandra

Package in package 'Class Diagram'

Cassandra
Version 1.0 Phase 1.0 Proposed
df-carvalho created on 04/12/2020. Last modified 04/12/2020

Starter Class Diagram

Package in package 'Cassandra'

Starter Class Diagram
Version 1.0 Phase 1.0 Proposed
df-carvalho created on 04/12/2020. Last modified 23/01/2019

Class Diagram diagram

Class diagram in package 'Starter Class Diagram'

Class Diagram
Version 1.0
df-carvalho created on 04/12/2020. Last modified 06/12/2020

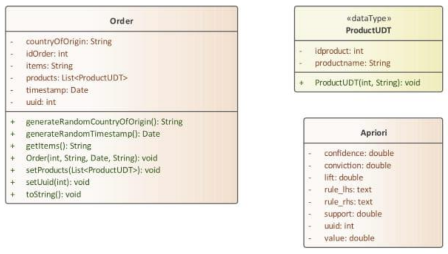
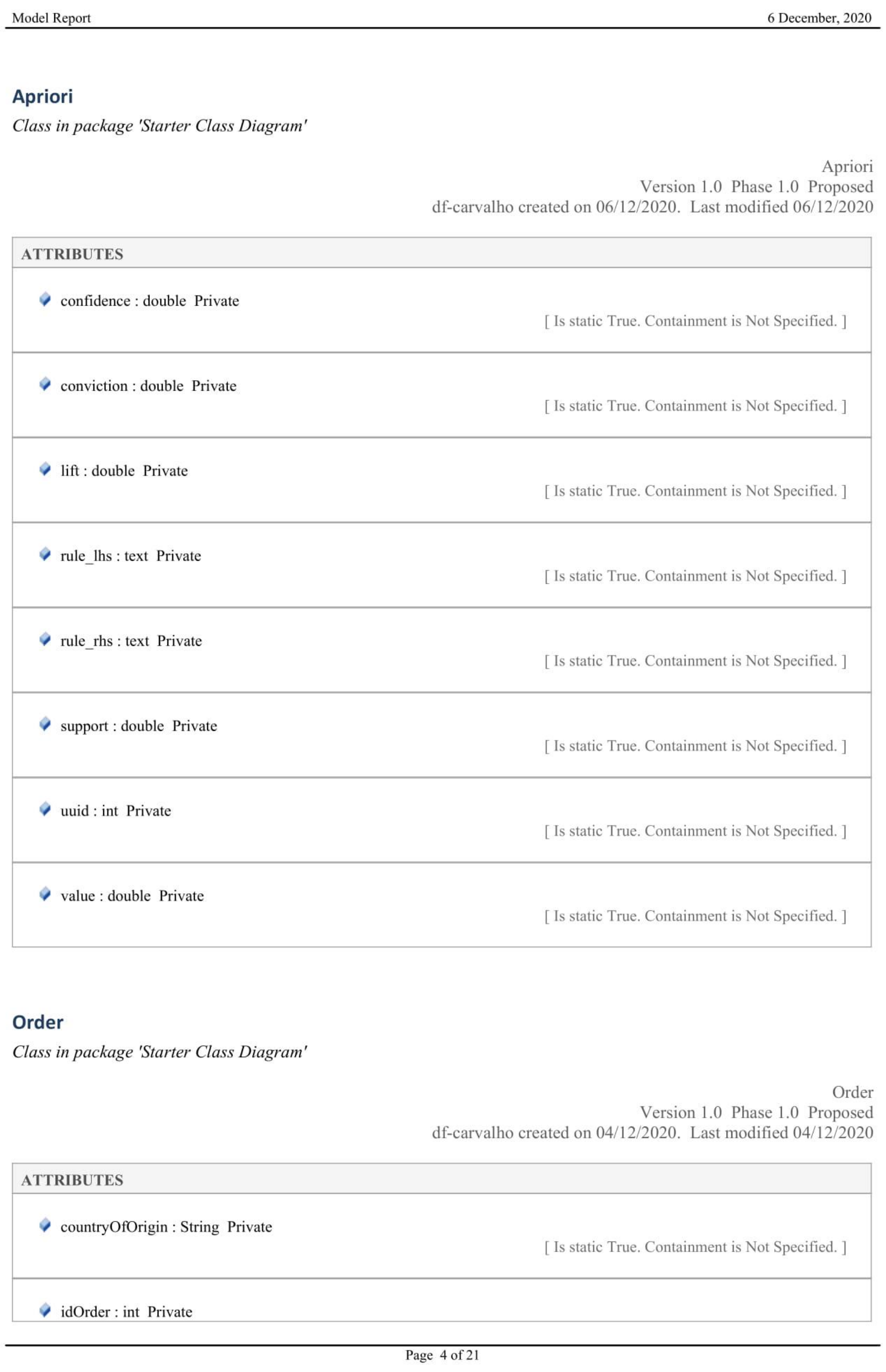


Figure 1: Class Diagram



Apriori

Class in package 'Starter Class Diagram'

Apriori
Version 1.0 Phase 1.0 Proposed
df-carvalho created on 06/12/2020. Last modified 06/12/2020

ATTRIBUTES
confidence : double Private [Is static True. Containment is Not Specified.]
conviction : double Private [Is static True. Containment is Not Specified.]
lift : double Private [Is static True. Containment is Not Specified.]
rule_lhs : text Private [Is static True. Containment is Not Specified.]
rule_rhs : text Private [Is static True. Containment is Not Specified.]
support : double Private [Is static True. Containment is Not Specified.]
uuid : int Private [Is static True. Containment is Not Specified.]
value : double Private [Is static True. Containment is Not Specified.]

Order

Class in package 'Starter Class Diagram'

Order
Version 1.0 Phase 1.0 Proposed
df-carvalho created on 04/12/2020. Last modified 04/12/2020

ATTRIBUTES
countryOfOrigin : String Private [Is static True. Containment is Not Specified.]
idOrder : int Private

ATTRIBUTES

[Is static True. Containment is Not Specified.]

items : String Private
[Is static True. Containment is Not Specified.]

products : List<ProductUDT> Private
[Is static True. Containment is Not Specified.]

timestamp : Date Private
[Is static True. Containment is Not Specified.]

uuid : int Private
[Is static True. Containment is Not Specified.]

OPERATIONS

generateRandomCountryOfOrigin () : String Public
[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

generateRandomTimestamp () : Date Public
[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

getItems () : String Public
[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

Order (idOrder : int , items : String , timestamp : Date , countryOfOrigin : String) : void Public
[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

setProducts (products : List<ProductUDT>) : void Public
[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

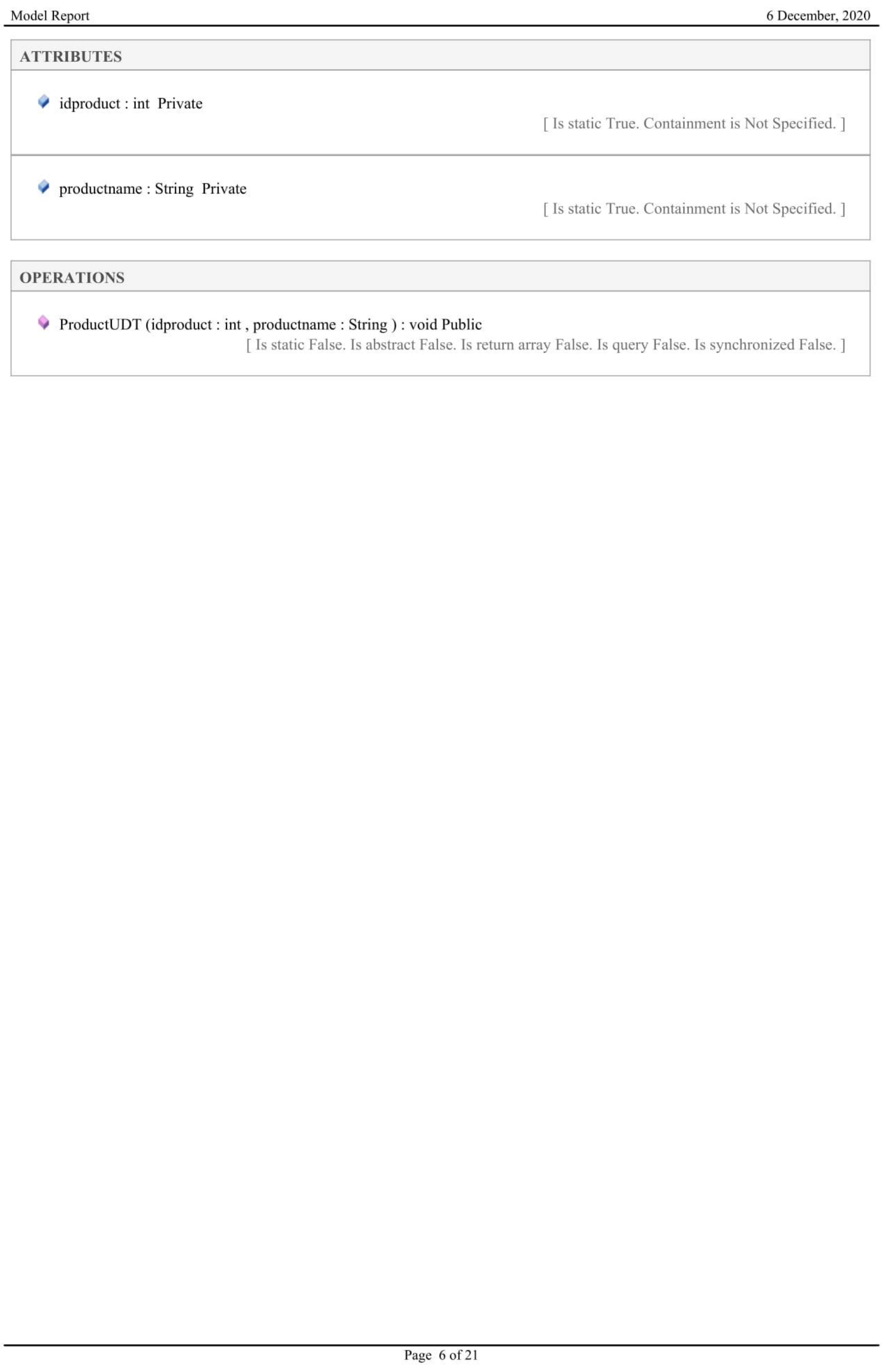
setUuid (uuid : int) : void Public
[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

toString () : void Public
[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

ProductUDT

DataType in package 'Starter Class Diagram'

ProductUDT
Version 1.0 Phase 1.0 Proposed
df-carvalho created on 04/12/2020. Last modified 04/12/2020



Model Report 6 December, 2020

ATTRIBUTES

◆ idproduct : int Private
[Is static True. Containment is Not Specified.]

◆ productname : String Private
[Is static True. Containment is Not Specified.]

OPERATIONS

◆ ProductUDT (idproduct : int , productname : String) : void Public
[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

Page 6 of 21

PostgreSQL

Package in package 'Class Diagram'

PostgreSQL
Version 1.0 Phase 1.0 Proposed
df-carvalho created on 04/12/2020. Last modified 04/12/2020

Starter Class Diagram

Package in package 'PostgreSQL'

Starter Class Diagram
Version 1.0 Phase 1.0 Proposed
df-carvalho created on 04/12/2020. Last modified 23/01/2019

Class Diagram diagram

Class diagram in package 'Starter Class Diagram'

Class Diagram
Version 1.0
df-carvalho created on 04/12/2020. Last modified 04/12/2020

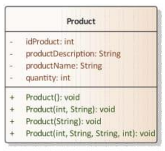


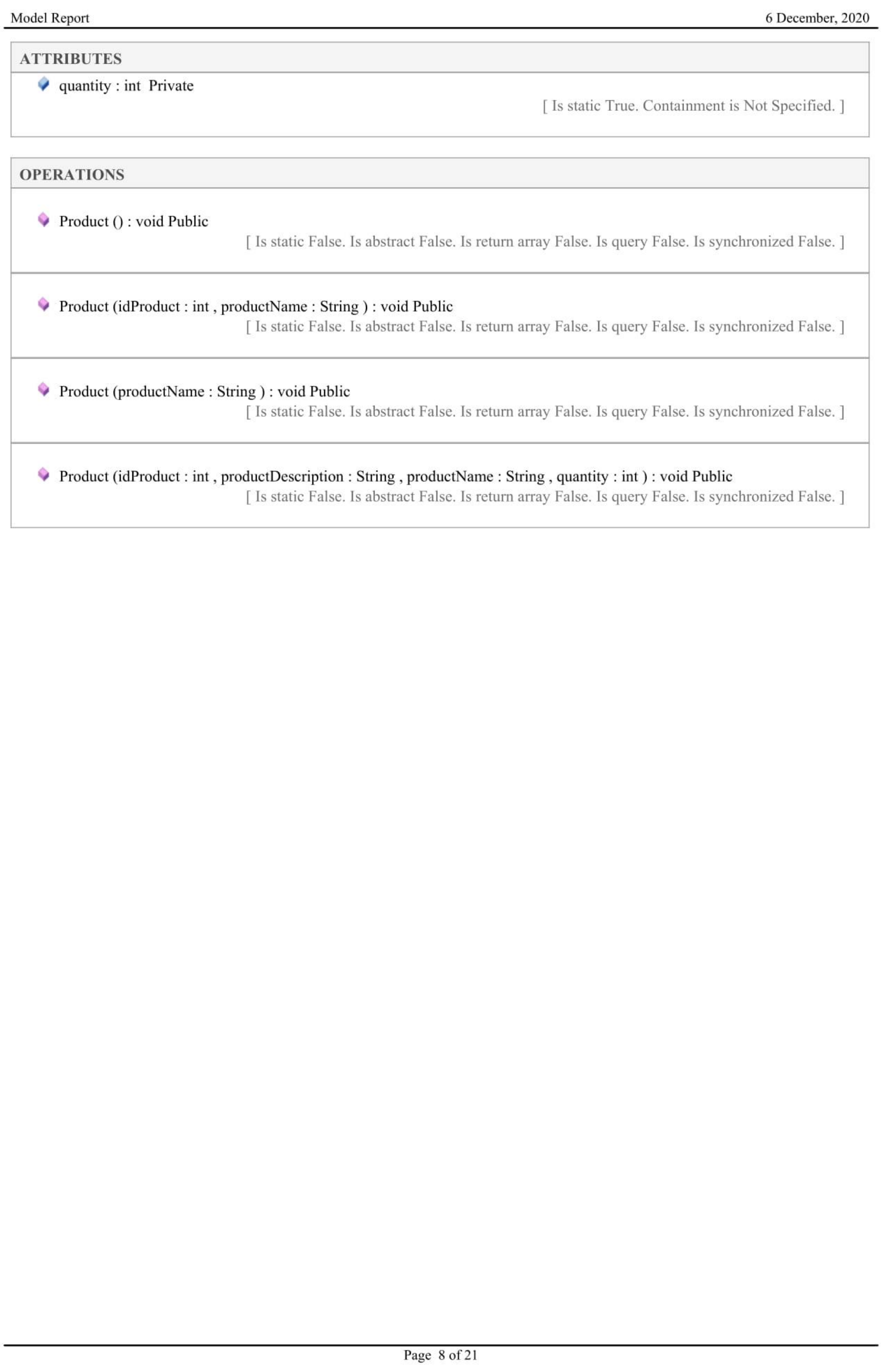
Figure 2: Class Diagram

Product

Class in package 'Starter Class Diagram'

Product
Version 1.0 Phase 1.0 Proposed
df-carvalho created on 04/12/2020. Last modified 04/12/2020

ATTRIBUTES
idProduct : int Private [Is static True. Containment is Not Specified.]
productDescription : String Private [Is static True. Containment is Not Specified.]
productName : String Private [Is static True. Containment is Not Specified.]



ATTRIBUTES
quantity : int Private [Is static True. Containment is Not Specified.]

OPERATIONS
Product () : void Public [Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]
Product (idProduct : int , productName : String) : void Public [Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]
Product (productName : String) : void Public [Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]
Product (idProduct : int , productDescription : String , productName : String , quantity : int) : void Public [Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

Component Diagram

Package in package 'Model'

Component Diagram
Version 1.0 Phase 1.0 Proposed
df-carvalho created on 04/12/2020. Last modified 05/12/2020

Starter Component Diagram diagram

Component diagram in package 'Component Diagram'

Starter Component Diagram
Version 1.0
df-carvalho created on 04/12/2020. Last modified 06/12/2020

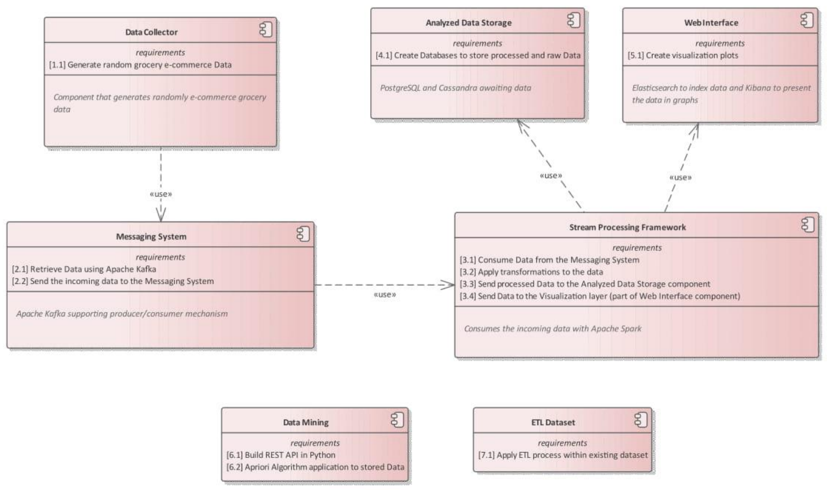

Data Collector
requirements
[1.1] Generate random grocery e-commerce Data
Component that generates randomly e-commerce grocery data
Analyzed Data Storage
requirements
[4.1] Create Databases to store processed and raw Data
PostgreSQL and Cassandra awaiting data
Web Interface
requirements
[5.1] Create visualization plots
Elasticsearch to index data and Kibana to present the data in graphs
«use»
«use»
«use»
Messaging System
requirements
[2.1] Retrieve Data using Apache Kafka
[2.2] Send the incoming data to the Messaging System
Apache Kafka supporting producer/consumer mechanism
«use»
Stream Processing Framework
requirements
[3.1] Consume Data from the Messaging System
[3.2] Apply transformations to the data
[3.3] Send processed Data to the Analyzed Data Storage component
[3.4] Send Data to the Visualization layer (part of Web Interface component)
Consumes the incoming data with Apache Spark
Data Mining
requirements
[6.1] Build REST API in Python
[6.2] Apriori Algorithm application to stored Data
ETL Dataset
requirements
[7.1] Apply ETL process within existing dataset

Figure 3: Starter Component Diagram

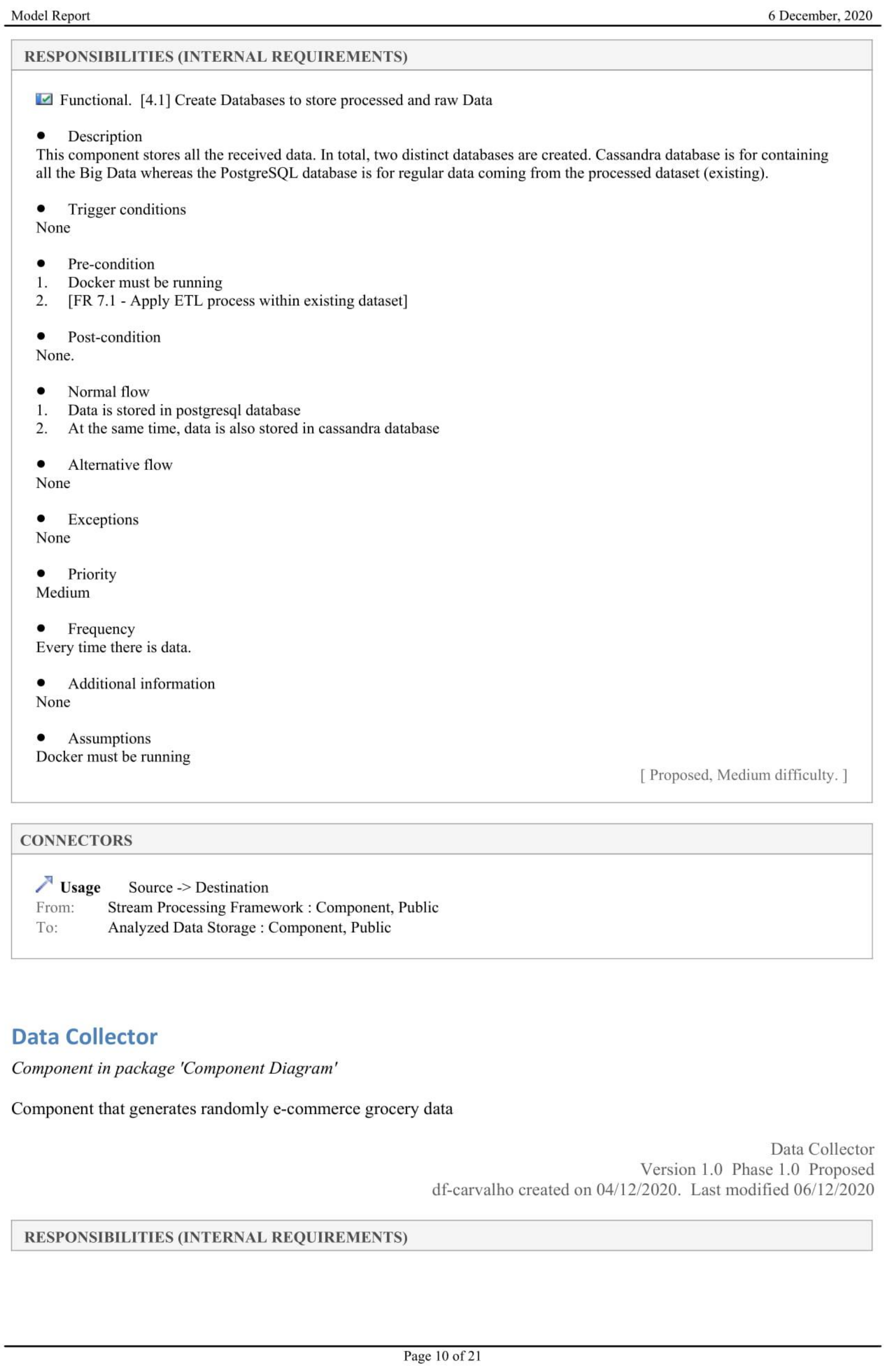
Analyzed Data Storage

Component in package 'Component Diagram'

PostgreSQL and Cassandra awaiting data

Analyzed Data Storage
Version 1.0 Phase 1.0 Proposed
df-carvalho created on 04/12/2020. Last modified 06/12/2020

RESPONSIBILITIES (INTERNAL REQUIREMENTS)



Model Report 6 December, 2020

RESPONSIBILITIES (INTERNAL REQUIREMENTS)

Functional. [4.1] Create Databases to store processed and raw Data

- Description

This component stores all the received data. In total, two distinct databases are created. Cassandra database is for containing all the Big Data whereas the PostgreSQL database is for regular data coming from the processed dataset (existing).

- Trigger conditions

None

- Pre-condition

1. Docker must be running
2. [FR 7.1 - Apply ETL process within existing dataset]

- Post-condition

None.

- Normal flow

1. Data is stored in postgresql database
2. At the same time, data is also stored in cassandra database

- Alternative flow

None

- Exceptions

None

- Priority

Medium

- Frequency

Every time there is data.

- Additional information

None

- Assumptions

Docker must be running

[Proposed, Medium difficulty.]

CONNECTORS

Usage Source -> Destination

From:	Stream Processing Framework : Component, Public
To:	Analyzed Data Storage : Component, Public

Data Collector

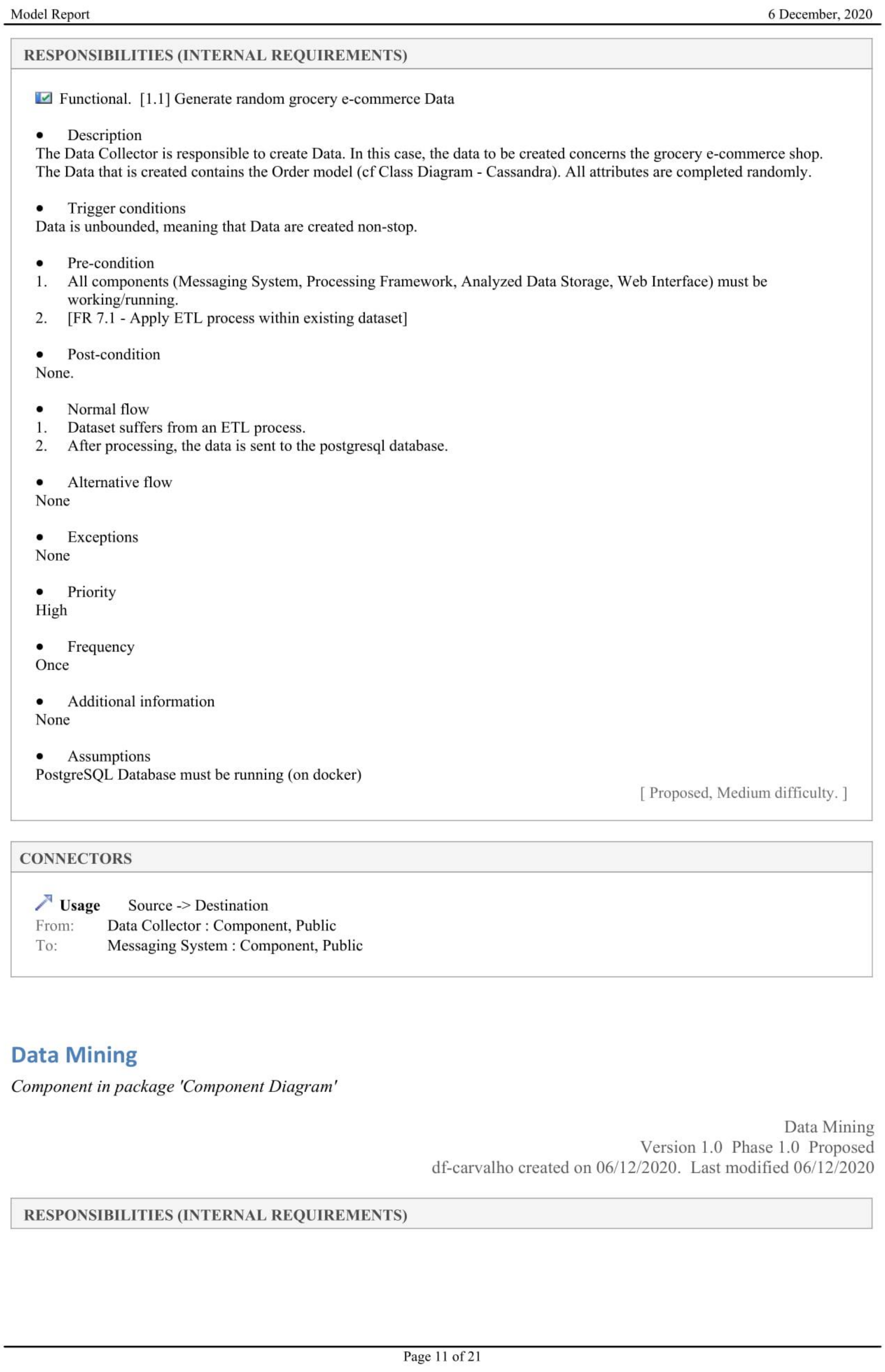
Component in package 'Component Diagram'

Component that generates randomly e-commerce grocery data

Data Collector
Version 1.0 Phase 1.0 Proposed
df-carvalho created on 04/12/2020. Last modified 06/12/2020

RESPONSIBILITIES (INTERNAL REQUIREMENTS)

Page 10 of 21



RESPONSIBILITIES (INTERNAL REQUIREMENTS)

Functional. [1.1] Generate random grocery e-commerce Data

- Description

The Data Collector is responsible to create Data. In this case, the data to be created concerns the grocery e-commerce shop. The Data that is created contains the Order model (cf Class Diagram - Cassandra). All attributes are completed randomly.

- Trigger conditions

Data is unbounded, meaning that Data are created non-stop.

- Pre-condition

1. All components (Messaging System, Processing Framework, Analyzed Data Storage, Web Interface) must be working/running.
2. [FR 7.1 - Apply ETL process within existing dataset]

- Post-condition

None.

- Normal flow

1. Dataset suffers from an ETL process.
2. After processing, the data is sent to the postgresql database.

- Alternative flow

None

- Exceptions

None

- Priority

High

- Frequency

Once

- Additional information

None

- Assumptions

PostgreSQL Database must be running (on docker)

[Proposed, Medium difficulty.]

CONNECTORS

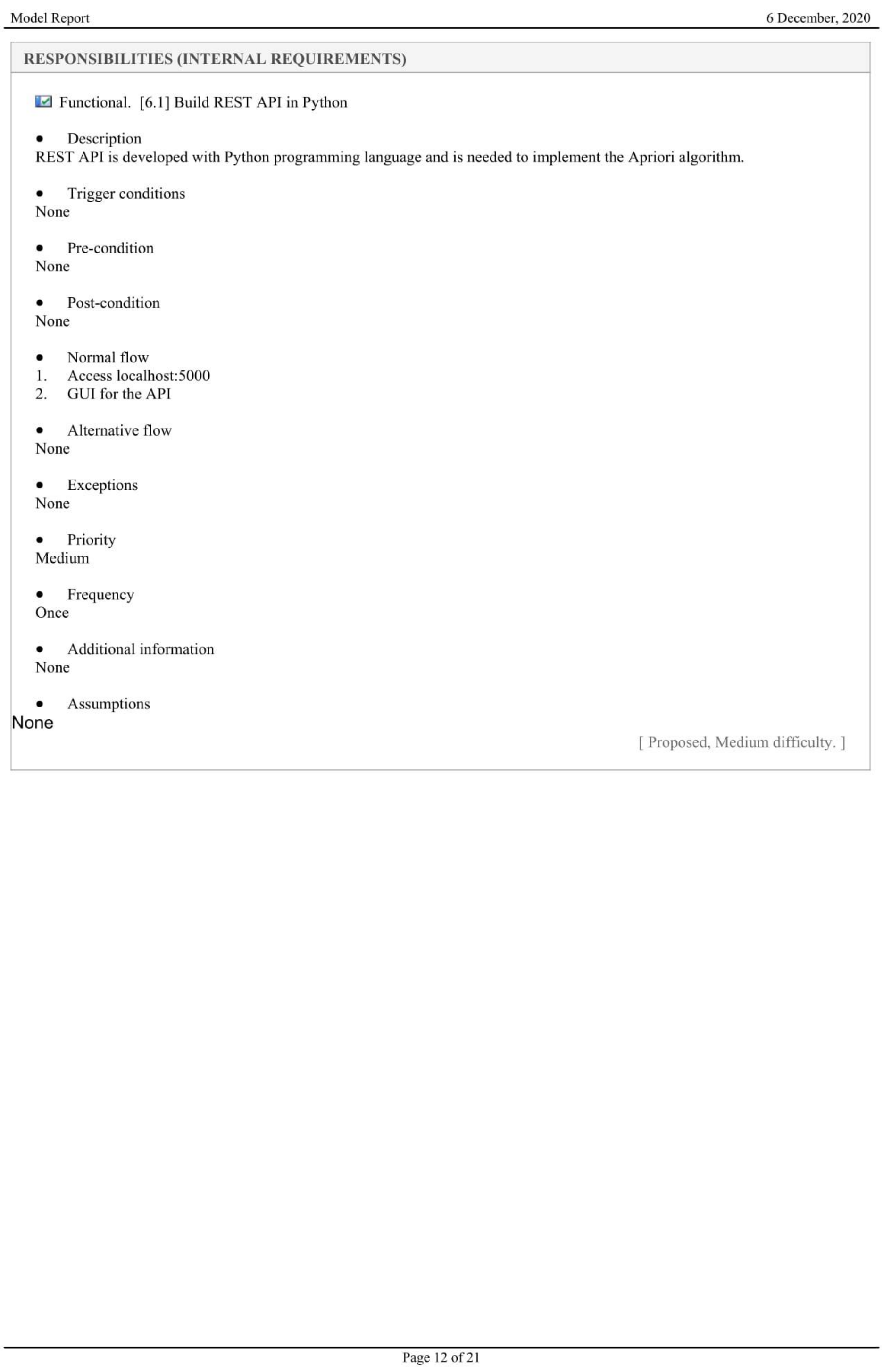
Usage Source -> Destination
From: Data Collector : Component, Public
To: Messaging System : Component, Public

Data Mining

Component in package 'Component Diagram'

Data Mining
Version 1.0 Phase 1.0 Proposed
df-carvalho created on 06/12/2020. Last modified 06/12/2020

RESPONSIBILITIES (INTERNAL REQUIREMENTS)



Model Report 6 December, 2020

RESPONSIBILITIES (INTERNAL REQUIREMENTS)

Functional. [6.1] Build REST API in Python

- Description

REST API is developed with Python programming language and is needed to implement the Apriori algorithm.

- Trigger conditions

None

- Pre-condition

None

- Post-condition

None

- Normal flow

1. Access localhost:5000
2. GUI for the API

- Alternative flow

None

- Exceptions

None

- Priority

Medium

- Frequency

Once

- Additional information

None

- Assumptions

None

[Proposed, Medium difficulty.]

Page 12 of 21

RESPONSIBILITIES (INTERNAL REQUIREMENTS)

Functional. [6.2] Apriori Algorithm application to stored Data

- Description

REST API method is implemented with the *Apriori* algorithm.

- Trigger conditions

None

- Pre-condition

None

- Post-condition

None

- Normal flow

1. Process data in cassandra database
2. Apply *Apriori* algorithm to the data
3. Store the data in one of cassandra databases

- Alternative flow

None

- Exceptions

None

- Priority

Medium

- Frequency

Every time is needed to reproduce apriori results

- Additional information

None

- Assumptions

None

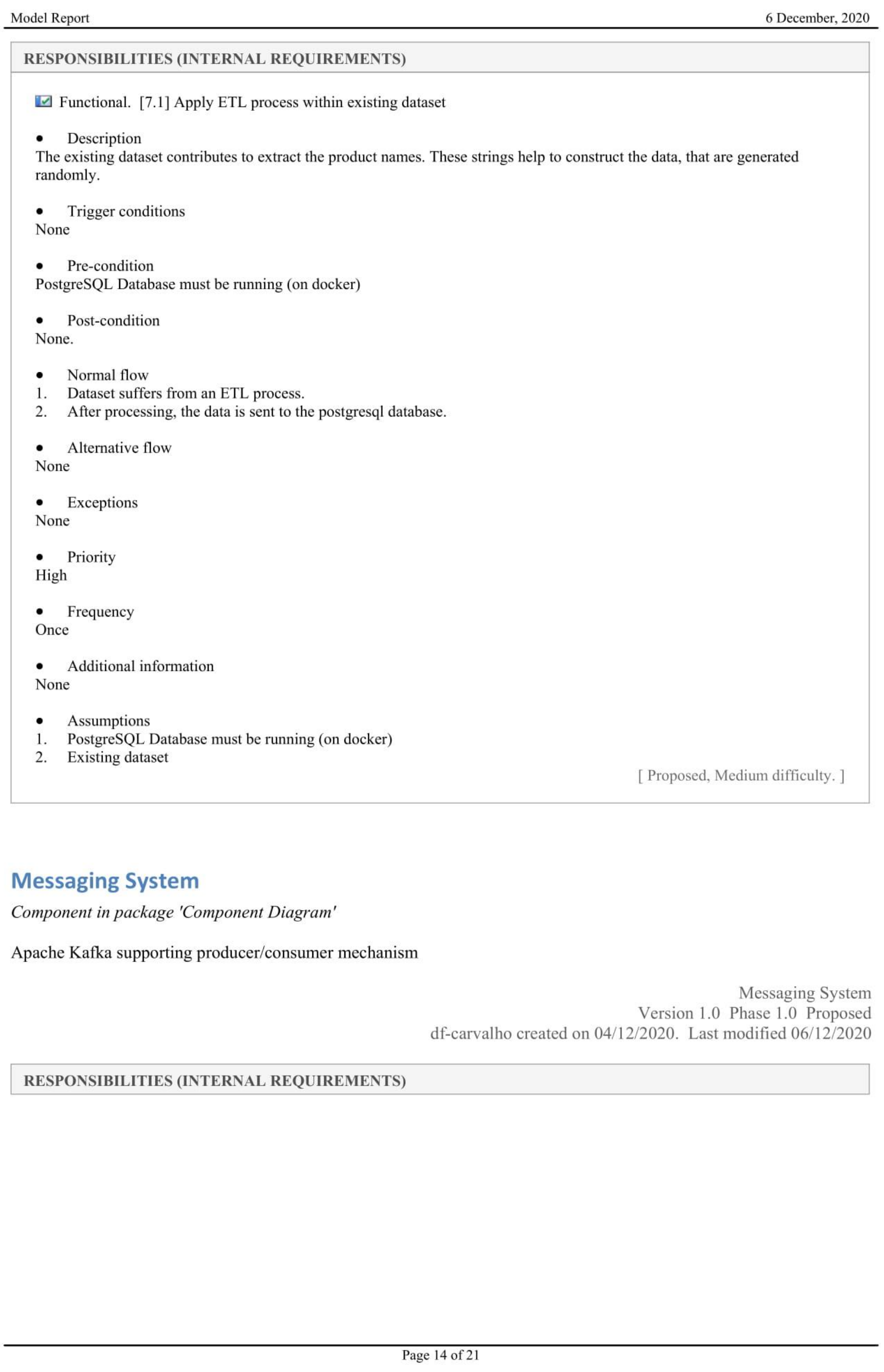
[Proposed, Medium difficulty.]

ETL Dataset

Component in package 'Component Diagram'

ETL Dataset
Version 1.0 Phase 1.0 Proposed
df-carvalho created on 06/12/2020. Last modified 06/12/2020

RESPONSIBILITIES (INTERNAL REQUIREMENTS)



RESPONSIBILITIES (INTERNAL REQUIREMENTS)

Functional. [7.1] Apply ETL process within existing dataset

- Description

The existing dataset contributes to extract the product names. These strings help to construct the data, that are generated randomly.

- Trigger conditions

None

- Pre-condition

PostgreSQL Database must be running (on docker)

- Post-condition

None.

- Normal flow

1. Dataset suffers from an ETL process.
2. After processing, the data is sent to the postgresql database.

- Alternative flow

None

- Exceptions

None

- Priority

High

- Frequency

Once

- Additional information

None

- Assumptions

1. PostgreSQL Database must be running (on docker)
2. Existing dataset

[Proposed, Medium difficulty.]

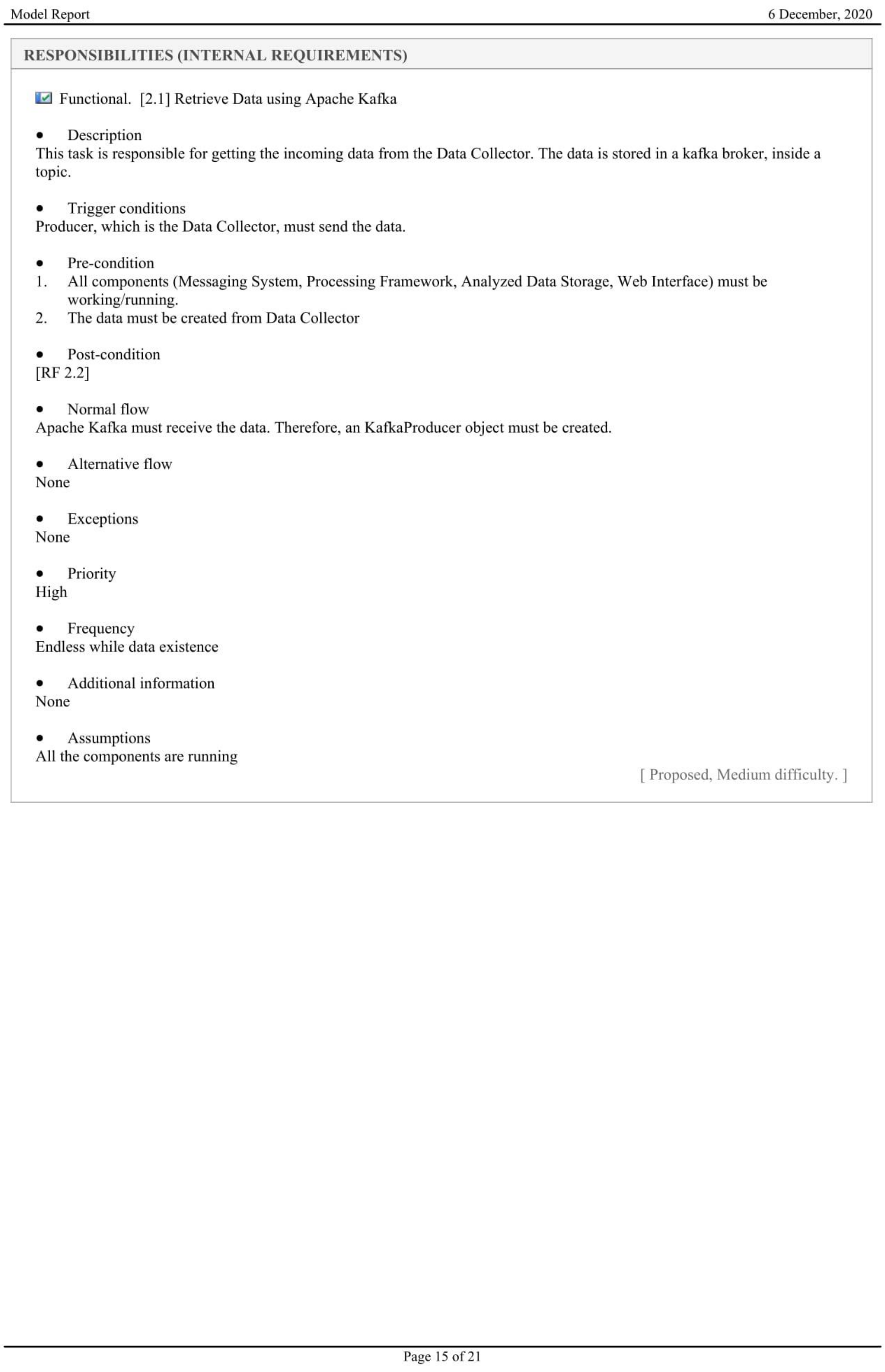
Messaging System

Component in package 'Component Diagram'

Apache Kafka supporting producer/consumer mechanism

Messaging System
Version 1.0 Phase 1.0 Proposed
df-carvalho created on 04/12/2020. Last modified 06/12/2020

RESPONSIBILITIES (INTERNAL REQUIREMENTS)



RESPONSIBILITIES (INTERNAL REQUIREMENTS)

✓ Functional. [2.1] Retrieve Data using Apache Kafka

- Description

This task is responsible for getting the incoming data from the Data Collector. The data is stored in a kafka broker, inside a topic.

- Trigger conditions

Producer, which is the Data Collector, must send the data.

- Pre-condition

1. All components (Messaging System, Processing Framework, Analyzed Data Storage, Web Interface) must be working/running.
2. The data must be created from Data Collector

- Post-condition

[RF 2.2]

- Normal flow

Apache Kafka must receive the data. Therefore, an KafkaProducer object must be created.

- Alternative flow

None

- Exceptions

None

- Priority

High

- Frequency

Endless while data existence

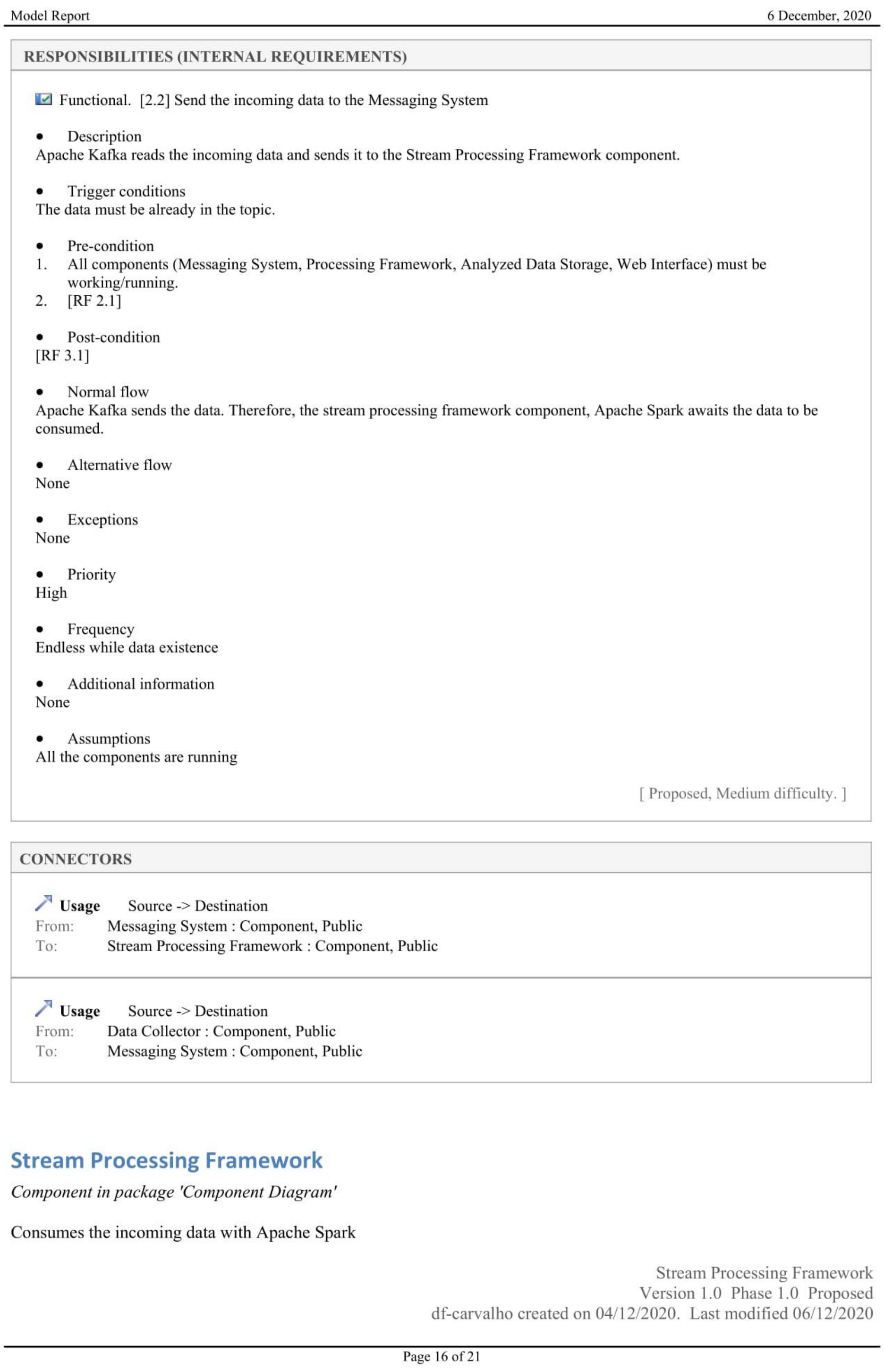
- Additional information

None

- Assumptions

All the components are running

[Proposed, Medium difficulty.]



RESPONSIBILITIES (INTERNAL REQUIREMENTS)

Functional. [2.2] Send the incoming data to the Messaging System

- Description

Apache Kafka reads the incoming data and sends it to the Stream Processing Framework component.

- Trigger conditions

The data must be already in the topic.

- Pre-condition

1. All components (Messaging System, Processing Framework, Analyzed Data Storage, Web Interface) must be working/running.
2. [RF 2.1]

- Post-condition

[RF 3.1]

- Normal flow

Apache Kafka sends the data. Therefore, the stream processing framework component, Apache Spark awaits the data to be consumed.

- Alternative flow

None

- Exceptions

None

- Priority

High

- Frequency

Endless while data existence

- Additional information

None

- Assumptions

All the components are running

[Proposed, Medium difficulty.]

CONNECTORS

Usage	Source -> Destination
From:	Messaging System : Component, Public
To:	Stream Processing Framework : Component, Public

Usage	Source -> Destination
From:	Data Collector : Component, Public
To:	Messaging System : Component, Public

Stream Processing Framework

Component in package 'Component Diagram'

Consumes the incoming data with Apache Spark

Stream Processing Framework
Version 1.0 Phase 1.0 Proposed
df-carvalho created on 04/12/2020. Last modified 06/12/2020

RESPONSIBILITIES (INTERNAL REQUIREMENTS)

✓ Functional. [3.1] Consume Data from the Messaging System

- Description

Stream processing framework reads the data from the topic

- Trigger conditions

- Pre-condition

1. All components (Messaging System, Processing Framework, Analyzed Data Storage, Web Interface) must be working/running.

- Post-condition

None.

- Normal flow

Data is read from the topic

- Alternative flow

None

- Exceptions

None

- Priority

High

- Frequency

Every time data is present

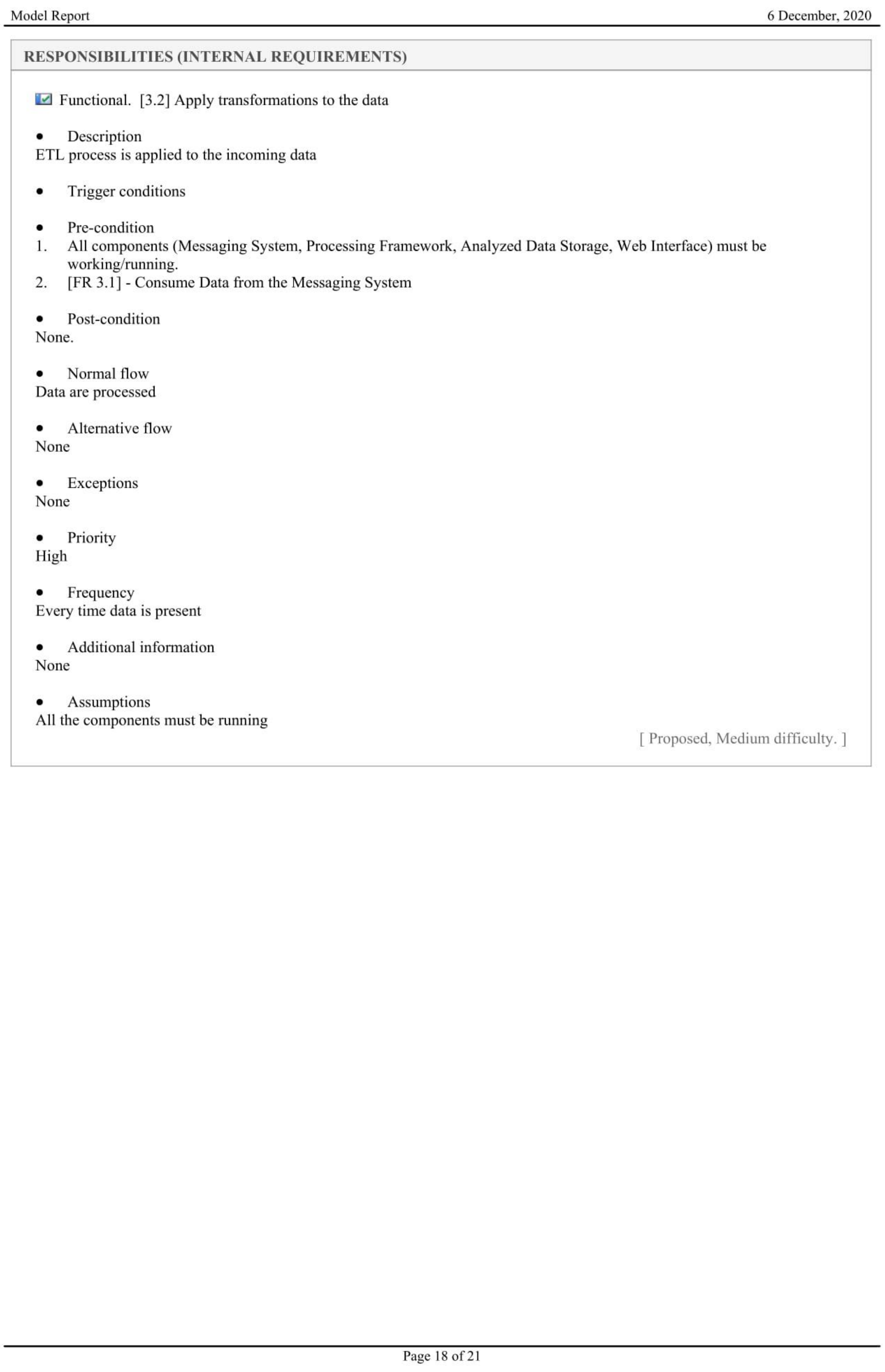
- Additional information

None

- Assumptions

All the components must be running

[Proposed, Medium difficulty.]



RESPONSIBILITIES (INTERNAL REQUIREMENTS)

Functional. [3.2] Apply transformations to the data

- Description

ETL process is applied to the incoming data

- Trigger conditions

- Pre-condition

1. All components (Messaging System, Processing Framework, Analyzed Data Storage, Web Interface) must be working/running.
2. [FR 3.1] - Consume Data from the Messaging System

- Post-condition

None.

- Normal flow

Data are processed

- Alternative flow

None

- Exceptions

None

- Priority

High

- Frequency

Every time data is present

- Additional information

None

- Assumptions

All the components must be running

[Proposed, Medium difficulty.]

RESPONSIBILITIES (INTERNAL REQUIREMENTS)

✓ Functional. [3.3] Send processed Data to the Analyzed Data Storage component

- Description

Processed data is sent to the Data Storage component.

- Trigger conditions

- Pre-condition

1. All components (Messaging System, Processing Framework, Analyzed Data Storage, Web Interface) must be working/running.
2. [FR 3.2] - Apply transformations to the data

- Post-condition

None.

- Normal flow

Data is sent to the existing databases

- Alternative flow

None

- Exceptions

None

- Priority

Medium

- Frequency

Every time data is present

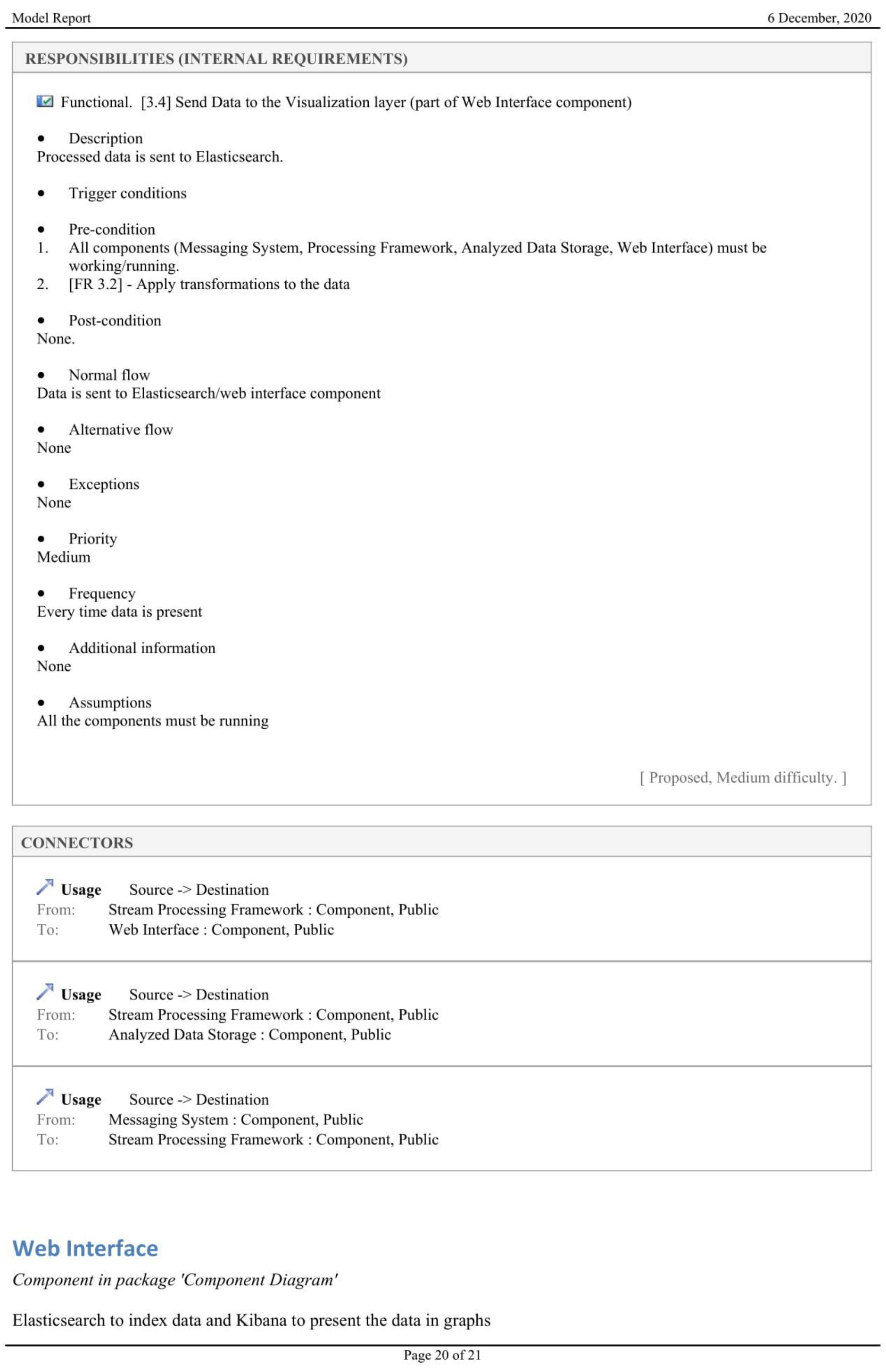
- Additional information

None

- Assumptions

All the components must be running

[Proposed, Medium difficulty.]



RESPONSIBILITIES (INTERNAL REQUIREMENTS)

Functional. [3.4] Send Data to the Visualization layer (part of Web Interface component)

- Description

Processed data is sent to Elasticsearch.

- Trigger conditions

- Pre-condition

1. All components (Messaging System, Processing Framework, Analyzed Data Storage, Web Interface) must be working/running.
2. [FR 3.2] - Apply transformations to the data

- Post-condition

None.

- Normal flow

Data is sent to Elasticsearch/web interface component

- Alternative flow

None

- Exceptions

None

- Priority

Medium

- Frequency

Every time data is present

- Additional information

None

- Assumptions

All the components must be running

[Proposed, Medium difficulty.]

CONNECTORS

Usage Source -> Destination
From: Stream Processing Framework : Component, Public
To: Web Interface : Component, Public

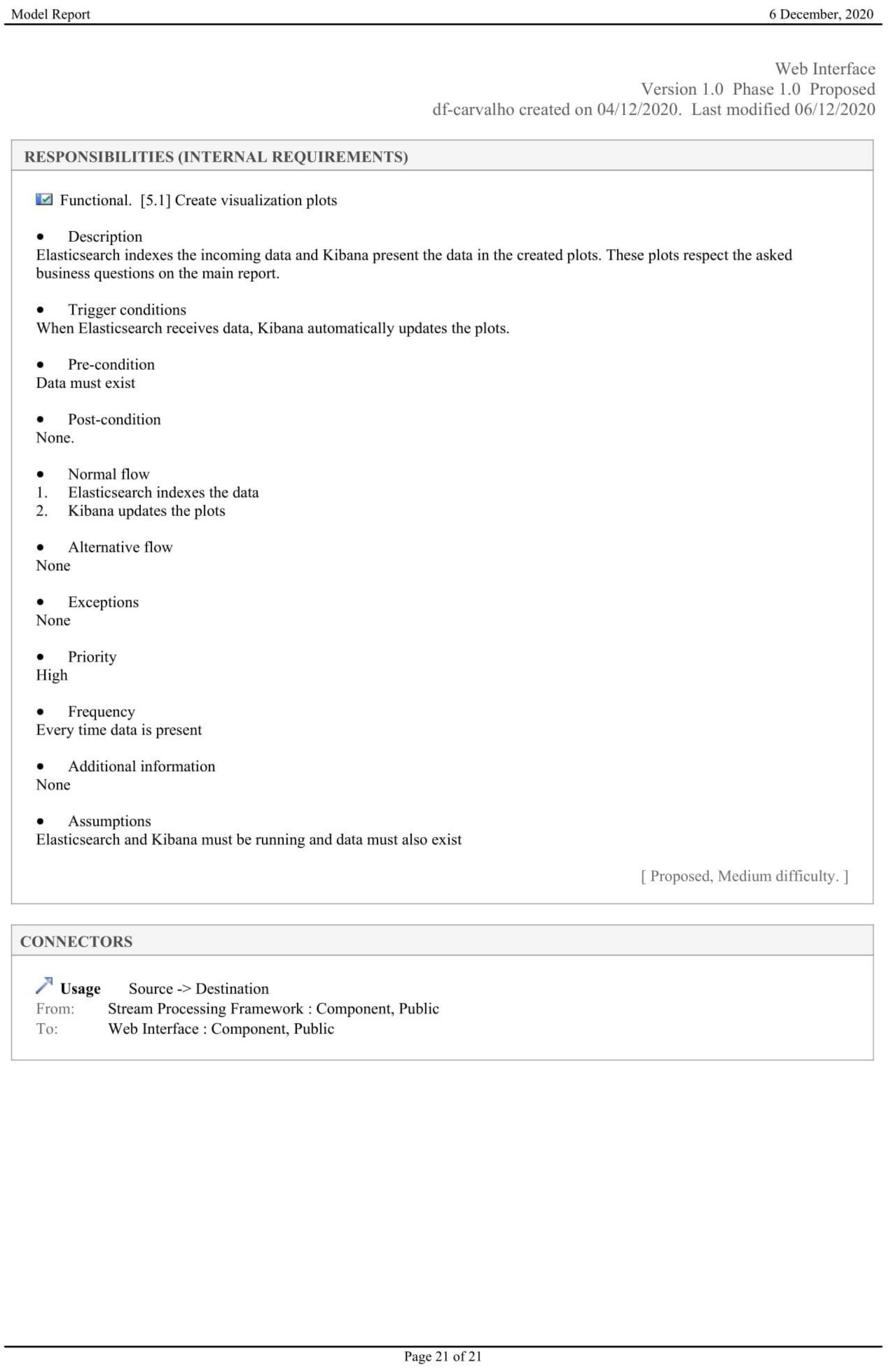
Usage Source -> Destination
From: Stream Processing Framework : Component, Public
To: Analyzed Data Storage : Component, Public

Usage Source -> Destination
From: Messaging System : Component, Public
To: Stream Processing Framework : Component, Public

Web Interface

Component in package 'Component Diagram'

Elasticsearch to index data and Kibana to present the data in graphs



Web Interface
Version 1.0 Phase 1.0 Proposed
df-carvalho created on 04/12/2020. Last modified 06/12/2020

RESPONSIBILITIES (INTERNAL REQUIREMENTS)

Functional. [5.1] Create visualization plots

- Description

Elasticsearch indexes the incoming data and Kibana present the data in the created plots. These plots respect the asked business questions on the main report.

- Trigger conditions

When Elasticsearch receives data, Kibana automatically updates the plots.

- Pre-condition

Data must exist

- Post-condition

None.

- Normal flow

1. Elasticsearch indexes the data
2. Kibana updates the plots

- Alternative flow

None

- Exceptions

None

- Priority

High

- Frequency

Every time data is present

- Additional information

None

- Assumptions

Elasticsearch and Kibana must be running and data must also exist

[Proposed, Medium difficulty.]

CONNECTORS

Usage Source -> Destination

From:	Stream Processing Framework : Component, Public
To:	Web Interface : Component, Public