Micro HealthCare Service Platform - Uma Solução Reativa Custo-Eficiente

HUGO ALEXANDRE FERNANDES DA SILVA TIGRE

Outubro de 2017

Instituto Superior de Engenharia do Porto

Micro HealthCare Service Platform

A Cost Effective Reactive Solution

Hugo Alexandre Fernandes da Silva Tigre

Dissertação para obtenção do Grau de Mestre em Engenharia Informática, Área de Especialização em Engenharia de Software

Advisor: Professor Especialista António Rocha

Porto, Outubro 2017

Resumo

O mercado farmacêutico, nos último anos, tem vindo a sofrer um aumento significativo de complexidade na gestão e implementação dos seus processos, resultado de um continuo aparecimento de novos serviços e informatização dos serviços existente, ambos os casos impostos por legislação. Para agravar a situação, este mercado está a passar um período de crise devido a cortes significativos nas margens de lucro das farmácias, resultado de medidas impostas pelo governo.

Este trabalho apresenta o desenvolvimento de uma plataforma digital de serviços que pretende responder às necessidades de unificação de serviços, e desta forma reduzir a complexidade dos mesmos, e ao mesmo tempo manter a produção e manutenção da plataforma a um baixo custo sem comprometer a eficiência e utilidade da mesma.

Esta plataforma foi desenvolvida fazendo uso de tecnologias de livre uso comercial e com resultados de efetividade comprovados. A combinação destas tecnologias e a arquitetura das suas implementações são o resultado da análise de requisitos funcionais e não funcionais e de uma grande atenção aos custos.

As tecnologias principais podem-se resumir, mas não se limitam, à Play Framework e a Akka Framework, estas duas são as principais responsáveis pela orientação reativa da plataforma.

A implementação desta plataforma foi testada tecnicamente, mas a sua praticabilidade foi avaliada através da perceção, sobre a mesma, dos seus *stakeholders*. Neste momento a plataforma está implementada em formato piloto e apenas no primeiro trimestre de 2018 é que será efetuada a transição para produção.

Palavras-chave: Plataforma Reativa, Micro-serviços, Saúde, Privacidade

Abstract

The pharmaceutical market, in the last couple of years, has seen a high increase in complexity of its processes, both business focused and legally mandatory ones, this also resulted in a heterogeneous system, where fulfilling a purpose means producing different solutions with the same purpose. It's also a time of financial crisis in this market, a result of severe profit margin cuts by the government.

This work presents a solution to the technology problem while still addressing the financial one with the development of a unifying service platform. The development main focus areas are on the business requirements of the pharmaceutical market to facilitate its day to day needs and provide a safer environment, on the reactive manifesto principles to respond to service availability requirements, cost effective implementation to better adapt to the market and simplicity to facilitate the continuous maintenance and evolution of the platform.

Keywords: Reactive Platform, Micro-services, Healthcare, Privacy

Acknowledgements

Above all, my thanks to my teacher and advisor Prof. Dr. António Rocha who always tried to directed me in the right path and to the educational institution in general for providing the means to achieve this goal.

To my family, for all the support throughout the years and providing me with the resources I needed to get where I am today, despite their sacrifices.

To my friends and dog, for just being there.

To all, my most sincere thank you.

Index

Figure Index

Table Index

Acronyms List

ACSS	Administração Central do Sistema de Saúde
AES	Advanced Encryption Standard
AFP	Associação de Farmácias de Portugal
ANF	Associação Nacional de Farmácias
BRMS	Business Rules Management System
BPMN	Business Process Model Notation
BPEL	Business Process Execution Language
CCF	Centro Conferência de Faturas
CQRS	Command Query Responsibility Segregation
CNPD	Comissão Nacional de Proteção de Dados
CSRF	Cross Site Request Forgery
CORS	Cross Origin Resource Sharing
DSL	Domain Specific Language
DDD	Domain-Driven Design
ESB	Enterprise Service Bus
EAI	Enterprise Application Integration
FQN	Full Qualified Name
HOCON	Human-Optimized Config Object Notation
HTTP	Hypertext Transfer Protocol
HL7	Health Level Seven International
HIV/AIDS	Human immune deficiency virus infection and acquired immune deficiency syndrome
ISDN	Integrated Service Digital Network
I/O	Input and Output
jBPM	Business Process Management

JSON	JavaScript Object Notation
JWT	JSON Web Token
LE	Large Enterprise
MHP	Micro Healthcare Platform
MVP	Minimum Viable Product
MVC	Model View Controller
ODE	Orchestration Director Engine
ODM	Operational Decision Manager
OO	Object Oriented
OOP	Object Oriented Programming
OWASP	Open Web Application Security Project
PBS	Product Breakdown Structure
REST	Representational State Transfer
SME	Small Medium Enterprise
SOA	Service-Oriented Architecture
SPMS	Serviços Partilhados do Ministério da Saúde
SNS	Serviço Nacional de Saúde
SNI	Server Name Indication
SPA	Single Page App
SBT	Simple Build Tool
SCA	Service Component Architecture
TARV / HAART	Highly Active Antiretroviral Therapy
TLS	Transport Layer Security
TPA	Terminal de Pagamento Automático (Credit Card POS Terminal)
XSS	Cross Site Scripting
XOP	XML-binary Optimized Packaging

WSS	Secure WebSockets

1. Introduction

This chapter presents an introduction to this work context, problems, intended solutions as well as the proposed value of this project.

1.1. Problem and Context

The health care industry has evolved in the last couple of years, although, not in a uniform way, maybe because it's dependency on government funds, high level of complexity, data sensitivity and, specially, government parties that dictate changes without understanding all its technical side effects, all this, resulting in premature systems update and scattered flaws in the flow/exchange of information. The result is a sector with several characteristic flaws, like several product order protocols[1], all currently active in the same market, with the exact same purpose but, technologically, very different from one another.

The heterogeneous nature of existing systems from different platforms, business and languages needs to be integrated in a more seamless way and communicate more reliably (LI et al., 2012). There are high security concerns when user healthcare data is concern, especially, when in the past, user data has been lost or stolen, some cases might even go undetected which damaging effects are not always clear or measurable (Kang et al., 2016).

This proposes the need to create better means to serve the healthcare market and all participating parties, so, is the objective of this thesis to plan and develop a Healthcare Enterprise Service Platform (MHP), with main focus on simplicity, lightweight, replicability, scalability and that follow the reactive manifesto principles[2].

1 Currently active order protocol include a protocol from 1994 that all pharmacies use and several suppliers have their own protocols that are used by just their clients.

2 https://www.reactivemanifesto.org

The healthcare market is a big market and involves a lot of different business and institutions, this project will focus more on the pharmaceutical market and the needs of its more close professional relations, like suppliers (products and information), laboratories, clients and pharmacy associations.

The following is a list of issues this project addresses:

- Diversity and complexity in services provided;
- Diversity of relational entities, sometimes with conflicting interests, resulting in not providing sufficient incentive in the share of information and knowledge;
- Entity size difference, multinationals, small local companies, with different technologies and know-how, resulting in bad interoperability and loss of information;
- Need to better protect sensitive data, regulated by a national data protection committee agency (*Comissão Nacional de Proteção de Dados*)[3];
- Obsolete communication technologies that can't keep up with today's needs:
    - Ex: Pharmacies that make their orders with a protocol from 1994;
- Industry professionals needing access to data scattered from many sources;
- Non-existent practical cost effective platform for business.

1.2. Objectives

The time for this project to be developed is now, a consequence of the state of the art in the healthcare market, more precisely the pharmaceutical market, the more this kind of work is postponed the more complicated the market will become, this is shown by the constant appearances of new services, all of them very inconsistently, technology wise, related to one another[4].

This project objectives are focused on the state of the art of its development but there is no time of life for it, so, this work development and implementation, is thought in a way that facilitates its evolution and increased time of life.

The main objectives of this work are as follows:

a) All entities that communicate with this service(s) must be able to do it in a secure and cost-effective way:

3 www.cnpd.pt

4 As an example the recent program *"abem"* from Dignitude (www.dignitude.org/abem) has an invoice program which uses SOAP technologies and a prescription program that uses REST services, both are related and provided by the same company GLINT (*www.glintt.com*)

  - Meaning that security is mandatory, thus it will not exists services without security implemented, if the original services or protocol does not support security features, then it must be created a top layer with security features;
  - In this case, cost-effective wise means that services usage has a cost but it is always proportional with the usage;

b) Multi-Protocol support (some need to be supported from the beginning to increase this project initial value, including specific industry protocols that are currently essential to the industry):

  - REST and WebSockets;
  - Postal (Product Order from 1995), at the time developed for ISDN (Integrated Service Digital Network) networks;
  - *Via-Verde* Medication (www.infarmed.pt): protocol to control specific products orders and sales;
  - TARV (www.infarmed.pt): protocol to control specific product (HIV) availability;
  - New Orders and Products Information protocols, currently in development for this work to replace the old previous protocols;
  - Apostore (www.apostore.de), Technilab (www.tecnilab.com) and Consis (www.willach-pharmacy-solutions.com): product dispenser robot's protocols, this projects will proxy requests for dispense requests and orders checking;
  - Cashguard (www.cashguard.com): cash safe robot;
  - Associate cards validation: validation of associate cards that provide monetary cover for the medication cost. Initially there will be support for CGD (*Caixa Geral de Depósitos*), SNQTB (*Sindicato Nacional dos Quadros e Técnicos Bancários*) and SAVIDA from EDP (*Energias de Portugal*);
  - Pharmacy reimbursement Invoicing: All invoices emitted by the pharmacies to social and commercial entities that cover medication cost goes directly to the MHP and are, then, relayed to it's corresponding entity.

c) Data transformation: in order to support multiple protocols and serve has a technical facilitator for everyone this project needs to dedicate a lot of efforts into transforming data, for example, to accept a single request format for multiple provider protocols;

d) End-to-End Encryption (Entities using the MHP must be able to keep their data a secret form external and inside parties);

e) Uniform sharing of data: There's a lot of data that is used by all interested parties, which creates a need for a uniform sharing of this data, this is typically referred to has a dictionary data. For example, several entities use and share product information, but even with a unique identifier for each product, the name is rarely the exact same, this cases some problems, like in reporting;

f) Be agnostic of all entities using the MHP: The MHP should not treat differently, in any way, entities using the system, it's only concern should be the data, there is data more important than other, but all MHP clients should be treated as equals, meaning that all clients should receive the same quality of service and security, for example, if a client is abusing the systems resources, it's connection should not hinder other clients requests;

g) Document archiving and management: All messages that traverse the MHP should be saved for the longest time possible, forever if possible. Clients should have access to they messages/documents whenever they need to;

h) Scalable architecture: The MHP should be designed to scale, horizontally or vertically, very easily, without the need for additional development;

i) Create a solution always with simplicity in mind:

    - Independently of the requirements, the simplest solution possible should also be searched, even if this means that limitations are put in place in some services. This is to assure that this work lives on for the longest time possible;
    - If the architecture implementation is simple then it will also adapt easily to changes and it's easier to bring new professionals into the project;
    - Simplicity goes both ways, meaning technically and usability wise, usually, however, the effort to make technology implementation simple also helps in making it easier to use.

1.3. Stakeholder Concerns

Service mediators or platforms like Enterprise Service Bus (ESB) become popular because they enable flexible business collaboration, facilitating the intercommunication between business and the development of new services, the lack of efficient collaboration technologies has proven to have a negative impact on the exploitation of new business opportunities. Today, more and more, business are used to outsourcing for their technology needs, but the complexity of business logic and difficulty to adapt personnel to internal processes are usually barriers difficult to overcome (Lukác et al., 2016).

For this work to be accepted by all stakeholders, it must abide to some critical, less technical aspects:

- Proprietary development:
    - All main development will be done by the same company, but the technology and architecture will be open and documented, so that in the eventuality of it's founding company can no longer supported it, it, can be picked up by another company and/or developers;
- Consistency between services:
    - The architecture must be adapted to its needs, i.e., to the corresponding service, but must also respect the main architecture design, so has to maintain consistency between all the services in the MHP;
- Cost of development:
    - Initial development will have a higher cost, necessary to start up the project into its initial production phase, but this work should have in mind future work in the sense that adding new services, or changing old ones should be an easier process, ergo, less expensive one;
- Confidentiality:
    - There should be practices put in place that, not only are considered best practices security wise, but also, that give a sense of security to whom is using the services. The sense of security will facilitate this projects adoption.

1.4. Value Added

While defining the value of this work it's important to first understand the relationships between the stakeholders by creating a Value Network (see chapter 2.7.2), after which defining the perceived value benefits and sacrifices between this work developed product, the services it offers and its relationship with the stakeholders. This work proposes to fulfil some missing requirements of this network, by providing a commercial platform to uniform communication and data between the stakeholders while also providing a low cost and secure solution.

1.5. Expected Results

The results of this work can be divided in two main categories, one being the Minimum Viable Product (MVP) and the other being the Perceived Value from this project stakeholders.

1.5.1. Minimum Viable Product

Regarding the MVP, a Product Breakdown Structure (PBS) is used to give a top clear picture of what is intended (see Figure 10). PBS is a simplified hierarchy representation of this project structure, it divides the overall project into sub-projects and serves to reduce complexity giving a better overall view of the project (Hameri and Nitter, 2000). The top level represents the final product, while the rest are sub-components of the top component.

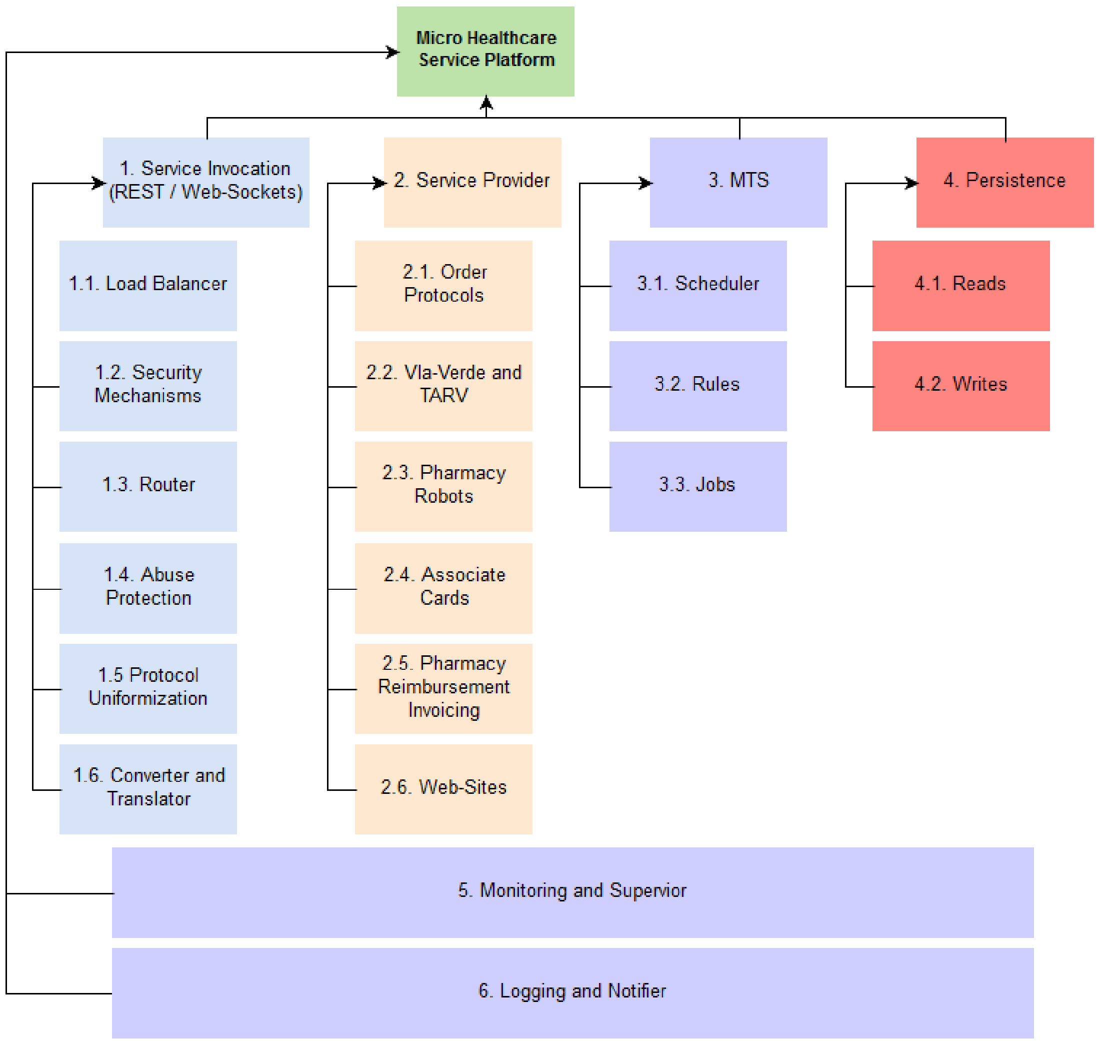


Figure 1: Product Breakdown Structure

1. **Service Invocation:** Supported protocols for service invocation are REST and WebSockets, SOAP is not supported in this project. This does not mean that the MHP does not communicate with SOAP services, it means that it does not respond to SOAP based requests.

    1.1. Load Balancer: All requests are load balanced to the requested service. The balance is achieved by analyzing the resources of the available instances of the backend service and redirecting the traffic to the one with the more resources available;

    1.2. Security Mechanisms: All request have to go through an authentication and authorization layer and only authorized traffic can reach the requested service;

    1.3. Router: The router is responsible for implementing the load-balancing and proxying the request. Besides the load-balancing the router acts as a API Gateway[5] for all provided services and also a Reverse Proxy[6] for supported web-sites that are inside and provided by this platform;

    1.4. Abuse Protection: Also implemented are Rate Limiting[7] and Circuit Breaker[8] techniques to avoid malicious users or bad scripts from overloading the system;

    1.5. Protocol Uniformization: To facilitate and promote this platform usage, the same will always provide a single service with the same business structure to interact will all providers whenever possible;

    1.6. Converter and Translator: this are the basic components for all service requests (see chapter 2.2);

2. **Service Provider:** This includes all the services already mentioned in chapter 1.2 and also to web-sites, giving them transparent HTTPS and load-balancing feature.

3. **MTS:** This component stands for Multi-Task Scheduler, its basically a multithreaded, concurrent scheduler that is responsible for processing jobs, which can be of two types, jobs that execute on-demand and jobs that are schedule to run at specific times or at specific conditions;

    3.1. Scheduler: responsible for controlling the schedule times of the jobs;

    3.2. Rules: these are the jobs rules, which are supposed to be edited by non-technical people, so it's configuration is not hard-coded;

5 https://martinfowler.com/articles/serverless.html
6 https://www.nginx.com/blog/building-microservices-using-an-api-gateway/ (NGINX is a popular Reverse Proxy software)
7 https://www.sans.org/reading-room/whitepapers/detection/denial-service-attacks-mitigation-techniques-real-time-implementation-detailed-analysi-33764 (pag.29)
8 https://martinfowler.com/bliki/CircuitBreaker.html

3.3. Jobs: the actual jobs implementation that this project supports;

4. **Persistence:** persistence is a major concern of the platform because it can be a source for performance degradation and it's usually more difficult to scale than the services. Because of this, in some cases, the reads will be physically separated from the writes.

5. **Monitoring and Supervisor:** The platform is a system that needs to keep components in communication with one another to control state. Monitoring implies that all components are monitored for their state and if intervention is needed, like stopping or restarting a service the Supervisors are responsible for acting upon their monitored children.

6. **Logging and Notifier:** this component is responsible for logging all messages, both service messages and debug messages, and notifying the developer team when errors occurs.

1.5.2. Perceived Value

The perceived value is not easy to measure, this is only possible after measuring specific categories perception for each stakeholder, i.e., for each stakeholder there are different categories to measure. This is done using Tony Lupo's approach as described in chapter 2.5.

To measure the perceived values for each category, an ordinal scale is used, so that all categories can be linearly ordered regarding the perceived quality and because there is no measure of distance between the choices (Flannelly et al., 2014) it makes this an easier scale for it's public, preventing random or incorrect values from lack of accuracy when filling in the questionnaires. Ordinal scales are also normally used in healthcare for measuring disease progression (Flannelly et al., 2014), which might make it more familiarly to some of this projects stakeholders.

A five choice scale, like the following figure, is used to keep it simple for the stakeholder, this way there is a better change of accuracy when comparing different stakeholder's opinions.

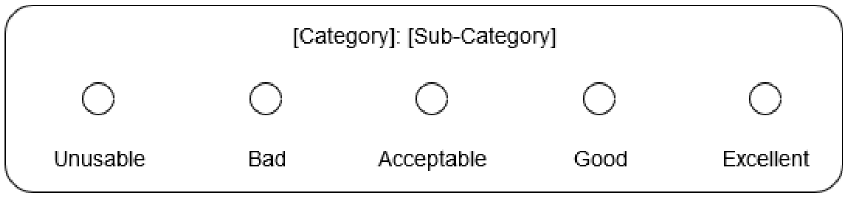


Figure 2: Perceived value scale

There is a scale, like the previous one, for each sub-category of each category identified in the quality categories diagram of chapter 2.5, these scales are distributed accordingly to every stakeholder. Because this project is in its early stages, the means of distributions of the scales are by email, followed by a phone contact if possible to increase the response ratio. If it doesn't exist direct contact with the stakeholder then the means of distribution will be through another stakeholder that has the means (for example: pharmacy patients need to be contacted through the pharmacy).

2. State of the Art

This chapter intends to provide a general concept of the state of the art in this project relevant technology and available solutions and/or approaches.

2.1. Business Opportunity

There's a lot of technology diversity in this market, meaning higher levels of complexity and information scattering. Simplifying technology integration and unifying healthcare data can provide better quality for all entities involved in this market.

Between mandatory services regulated by industry legislation, enforced by entities like *Serviços Partilhados do Ministério da Saúde*[9] (SPMS), Infarmed (National Medication and Health Products Authority) and *Administração Central do Sistema de Saúde* (ACSS), services required by pharmacy associations, *Associação Nacional das Farmácias*[10] (ANF) and *Associação de Farmácias de Portugal*[11] (AFP), and different implementations by the industry IT providers, including SPMS, the information flow is scattered between all these entities in different information collections.

By providing a unified platform for all these services and communication mechanisms to facilitate integration with this platform, value is added for anyone that needs or wants to use these services, for example, IT companies will have fewer, more secure and more standard protocols to implement, pharmacies will have better stability, higher data consistency and access to more services in its current business software applications and regulatory entities will have more adoption of its services and has an added facilitator, support request can be directed to one central entity.

9 www.spms.min-saude.pt

10 www.anfonline.pt

11 www.portaldasfarmacias.com

2.2. Principles and Methodologies

From a conceptual point of view this work provides a central platform that acts mainly as a mediator and translator between consumers and providers. This is, also, typically provided by an ESB, which underneath, is a service-oriented architecture (SOA), a distributed integration infrastructure which is message driven and provides routing and mediation services to interconnect senders and receivers (Li et al., 2012) and despite its business orientation, provides services to any number of clients, these can be suppliers, their clients, or any other kind of intermediary, providing the following key features:

1 Always available, service invocation with a service repository and message routing and storing, the clients only concern is to make a request (Ming-zhe, 2013). See "Router" and Transistor" bellow;

2 Heterogeneous system, the ESB has to provide message interaction between a predetermined number of protocols, typically this is done with web-service (Ming-zhe, 2013) (Lukác et al., 2016), but can be any other kind of message protocol. See "Converter" and "Translator" bellow;

3 Four essential functions: "Router" which transmits and routes messages according to content, "Converter" which transforms a communication protocol into another, "Translator" which transforms message format, dealing with business logic and "Transistor" which deals with business events (synchronous and asynchronous) from different services. Essentially "Converter" and "Translator" addresses service heterogeneous issues, while "Router" and "Transistor" addresses service reuse (Ming-zhe, 2013).

Many commercial and non-commercial products that exists today, claim to fulfil this requirements, but none of them completely fulfils the needs of both LE (Large Enterprises) and SME (Small Medium Enterprises). They need to be easy to understand and deploy, flexibly enough to adapt to different business needs and cheap enough to be adopted by small business (Lukác et al., 2016).

2.2.1. New Technologies

It's not necessary to innovate in new technologies to complete this project objectives / requirements, there already exists proven, useful and stable technologies for this project needs, however, there is more than one way to implement the same technology, and this is where the design considerations and decisions for this project take focus.

2.2.2. Reactive Manifesto

The high demand in responsiveness in today's systems requires a specific architecture that takes that into account, and steers the developer(s) into the right direction. This is where the Reactive Manifesto (Bonér et al., 2014) comes in. The manifesto states the main aspects that define what a Reactive System should be, which we could resume to a highly scalable and responsive system.

The Reactive Manifesto defines that a reactive application is based on four interrelated pillars: responsive, resilient, elastic and message driven (Bonér et al., 2014) (see Figure 3). An application is reactive if it is event-driven, able to provide an excellent user experience, able to better utilize the potential of the machines, and tolerate the mistakes and failures.

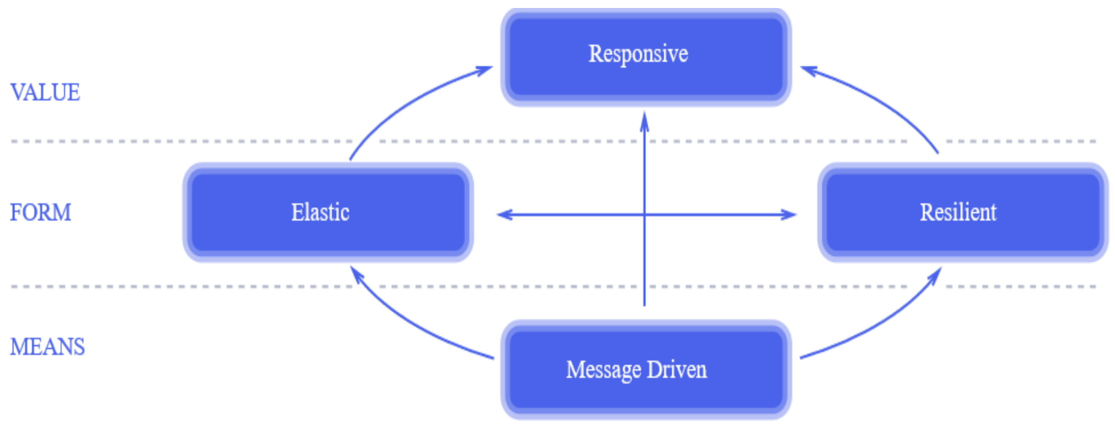


Figure 3: Reactive manifesto drivers (www.reactivemanifesto.org)

2.2.2.1. Responsive

Consistent quality of service is key here, response should be quick and errors should be handled effectively. The end result should encourage further interaction from the user. A responsive system is quick to respond to all requests in any situation (even in difficult situations) to ensure a good experience for its users (Bonér et al., 2014).

> “A message-driven architecture is very important to the responsiveness and provides an asynchronous boundary which decouples the user in time and space” (Mincer-Daszkiewicz, 2015).

2.2.2.2. Resilient

A resilient system, is one that remains responsive even when there is a failure, and it is achieved by guaranteeing high-availability (achieved by replicating when necessary), containment and isolation (if one component fails it should not compromise the system has a whole) and delegation (recovering of one component is done by another) (Bonér et al., 2014).

2.2.2.3. Elastic / Scalable

A scalable system in one that remains responsive under varying workload. Resiliency and scalability go hand-in-hand when creating consistently responsive applications. A scalable system is easily upgraded on demand in order to ensure responsiveness under various load conditions. The system should be able to scale as needed, for this, there should be no central bottleneck (Bonér et al., 2014).

2.2.2.4. Message Driven

A Message driven system is one that exchanges asynchronous messages between defined boundaries, loosening coupling between components and helping with isolation and location transparency (Bonér et al., 2014). The main difference between messages and events is that messages are directed while events happen. Messages have a clear destination while events may be observed by zero or more (0-N) observers (Halter and Shepherd, 2012).

2.2.2.5. Actor-based concurrency

Actor-based concurrency is an extension of the message-passing architecture, where messages are directed to a recipient, which happens to be an actor. Messages may cross thread boundaries or be passed to another actor's mailbox on a different physical server. This enables elasticity, scaling out on demand, as actors can be distributed across the network, yet still communicate with each other as if they were all sharing the same JVM (Halter and Shepherd, 2012).

2.2.2.6. Dealing with high number of requests

To better accept and adapt to a high number of requests requires the understanding and implementation of all the reactive manifesto principles. Message Driven can be seen has the means to achieve a reactive system, Elastic and Resilient has the form the system takes as it grows and Responsive has the value to the consumer[12].

2.2.3. Reactive Programming and Reactive Systems

When building micro-services, reactive programming plays a crucial role in working towards a reactive system, i.e., an efficient, responsive and highly scalable system. Developing should always mind the resources, keep the workflow asynchronous, minimizing contention and don't keep threads hostage. The following image serves has a clear picture of the effects in a system in a blocking vs non-blocking operations (Bonér 2017).

12 reactivemanifesto.org

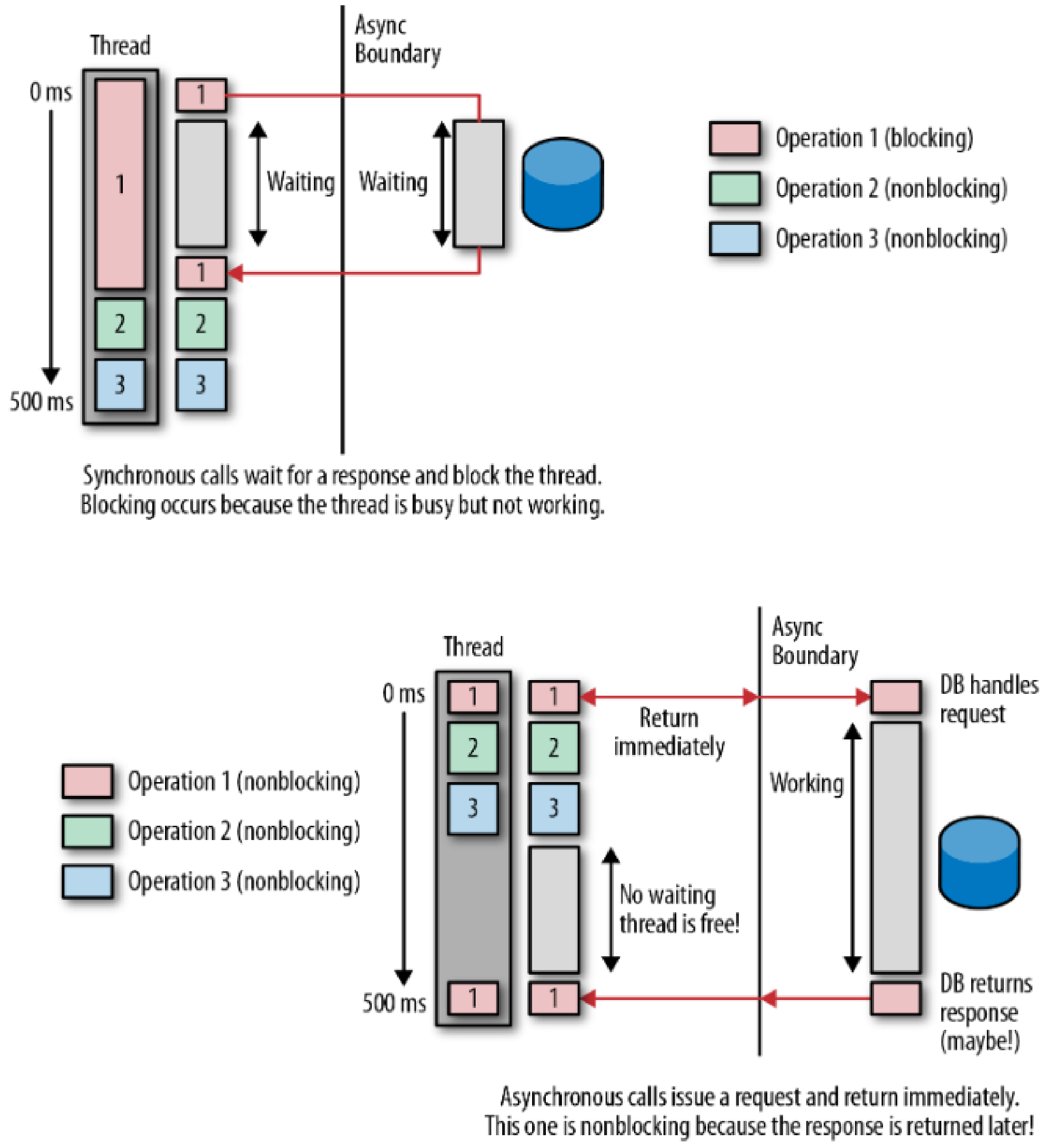

Thread
Async Boundary
0 ms
500 ms
1
2
3
Waiting
Waiting
Operation 1 (blocking)
Operation 2 (nonblocking)
Operation 3 (nonblocking)
Synchronous calls wait for a response and block the thread.
Blocking occurs because the thread is busy but not working.
Thread
Async Boundary
0 ms
500 ms
Return immediately
Working
No waiting thread is free!
DB handles request
DB returns response (maybe!)
Operation 1 (nonblocking)
Operation 2 (nonblocking)
Operation 3 (nonblocking)
Asynchronous calls issue a request and return immediately.
This one is nonblocking because the response is returned later!

Figure 4: Blocking vs Non-Blocking Operations (Bonér, 2017)

As it's possible to see in the picture, the non-blocking approach doesn't block the thread because it doesn't block while waiting for the database to return the result, instead, it assumes that a response may, or may not exist in the future.

2.2.3.1. Backpressure

Backpressure is the ability to control the pressure from requests, it can be applied in the front-end or the back-end part of a service, in this platform case the back-end is where it should be controlled. This is important as to avoid the services for overloading from some requests thus keeping the same from replying to other requests. The way this works, usually means that a backchannel exists to send small pieces of data into the upstream signalling if the communication flow should slow down and since both sides of the communication need to be able to understand this, it's important to have a standard which is being worked on right now with JDK9 *java.util.concurrent.Flow*[13] (Bonér, 2017).

Also In synchronous protocols, like REST, it's important to use back pressure, for example by sending a "server busy" HTTP message (503 status code), the client requesting the service can then interpret that it's time to slow down on the requests. This is usually accomplished with rate-limiting (see chapter 2.3.5.1).

2.2.3.2. Service Discovery

Service discovery is very important feature in a micro-service environment, without this it's not possible to transparently scale horizontally. Service addresses needs to be virtual in nature, meaning that a virtual address can point to one or more physical addresses, but from the point of view of configuration and development there is only one address. If a server fails, for example, the system needs to know this and send the message to one instance that is OK. This also allows for load-balancing capabilities (Bonér, 2017).

To achieve service discovery without static address a pattern called Inversion of Control[14] (IoC) Is used, this basically means that each service reports information back to the system to where it is and how it can be contacted. This information is saved and accessed using a Client-Side or Server-Side Service Registry[15] pattern.

2.2.4. Restrictions

As stated in chapter 1.1 this work focus on the pharmaceutical market and all its business relationships, which is just a component of the Healthcare market, however, the work being done is capable of evolving and being adapted to other use cases and requirements inside and outside the Healthcare market.

It's also the intention of this project to evolve past the objectives identified in Chapter 1.2, but this will not be considered until all these objectives are deemed stable by the stakeholders.

2.3. Technology

While this work focuses on some technologies to achieve its goals, it's important to be aware of the alternatives and the main advantages and disadvantages of the technologies available today.

13 http://www.reactive-streams.org

14 https://en.wikipedia.org/wiki/Inversion_of_control

15 http://microservices.io/patterns/service-registry.html

2.3.1. Message Formats

- **XML:** The more common data sharing mechanisms, such has those based on extensible markup language (XML) and simple object access protocol (SOAP), are a very time and processing consuming model which is directly proportional to the size and level of recurrence of the data transfers (Man et al., 2012), this, of course, compared to more lightweight protocols. There have been previously proposed techniques, like transferring all XML data in binary (XOP) (Gudgin et al., 2005), but this still was not optimal.

  Another major technology of this kind of system is the use of XSLT for the transformation of a message into another (Ming-zhe, 2013). XSLT processing is normally done with Saxon and/or Xalan. If XML is not optimal, then a protocol based on XML in which primary objective is to transform XML, will, also, never be optimal.

- **JSON:** JSON is a more lightweight (Simec et al., 2014) messaging format that accomplishes the same thing has XML, it doesn't have a schema, but it also doesn't suffer from schema related problems when integrating the message format into a platform and/or library.

  JSON is also a native format for NoSQL databases, like MongoDB and CouchDB.

- **HOCON:** HOCON[16] stand for Human-Optimized Config Object Notation, and this format was invented to facilitate human reading, it inherits its structure from JSON format and in the background it converts to JSON for interpretation and parsing, but reading and editing by a human is more convenient by the following characteristics:

  - Less noisy syntax, for example keys don't have to be encapsulated in quotes;
  - Ability to refer to another part of the document: for example, using a variable for repetitive values or using environment variables;
  - Its possible to concatenate different document files, i.e., importing a document into another;
  - It's possible to add comments to the document;
  - The syntax of the file can be more flat, like a Java system properties file;
  - For example, the following 3 examples are interpreted the same[17].

16 https://github.com/typesafehub/config/blob/master/HOCON.md
17 More examples at: https://github.com/typesafehub/config#examples-of-hocon

Table 1: HOCON Format

Example 1	Example 2	Example 3
	service {	service.auth.id = 2
	auth {	service {
service.auth.id = 2	id = 2	auth {
service.auth.name = auth	name = auth	name = auth
	}	}
	}	}

- **HL7:** HL7 is not really a format but a standard, as a format, it supports both XML and JSON message formats. HL7[18] was founded in 1987 as a non profit, and its mission is to provide standards for dealing with electronic healthcare information thus unifying the way the healthcare communicates and interoperates electronically (HL7, 2017).

  HL7 has well defined standards specifically for the healthcare market and it's free to use, supporting these standards by default could give more value to the platform. Implementing such support is not a simple task since the protocols can be quite extended, however, there are existing libraries that can help in this process and can be integrated in the platform, like for example the free and open source library HAPI[19].

2.3.2. REST Vs WebSockets

A typical RESTful HTTP connection is composed of the following steps:

1. Client creates a new TCP connection to the server;
2. Client and server negotiate using SSL handshake;
3. Client sends the request data, along with any headers;
4. Server sends response data to the client, along with any headers;
5. Client closes the connection.

If low latency is required then starting and closing connection for each request might not be an optimal solution because of the overhead of opening TCP connections and sending the same redundant header information multiple times (Gupta, 2014).

Since a RESTful connection has a strict order request and then response, and closes after that, the server is not able to push notification to the client. There is a "HTTP Keep Alive" optimization the can be used, but with limitations. The request and response order must be

18 http://www.hl7.org/
19 http://hl7api.sourceforge.net/

exact, meaning that multiples request cannot be send because the response to the second request might arrive earlier, also this does not reduce the header redundancy and has an approximately max timeout of 15 seconds, which from an API perspective is not very useful.

WebSockets on the other hand, are ideal for low latency connection, they can reuse the same connection to send multiple request and responses at the same time (full duplex), the timeout of the connection can be configured, there are headers in the connection but they only need to be sent one time per connection, the payload of each request is smaller (framed with 2 bytes) then the payload of a RESTful request (Gupta, 2014).

The next picture shows a comparison between a RESTful and a WebSocket connection, of the time to process a fix payload and an increasing number of messages. The performance increase of the WebSocket connection is proportional to the number of messages, this is because with REST, with every connection, a new TCP connection is established and new Headers are sent with the connection.

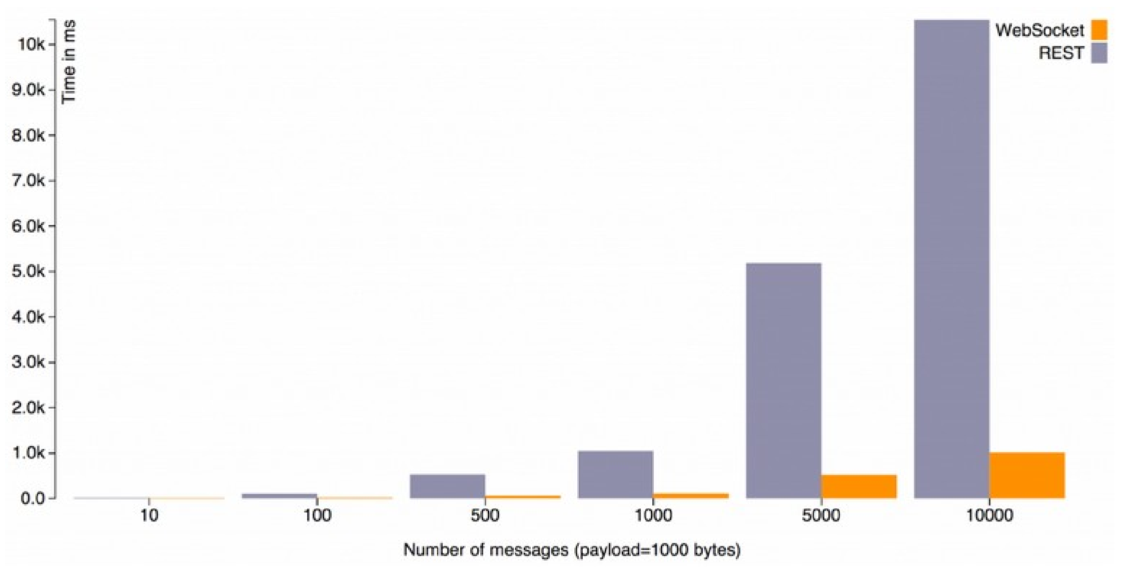


Figure 5: REST Vs WebSockets payload (Gupta 2014)

WebSockets can provide a big boost in performance in some cases, but there are still reasons to use REST architecture instead (Gupta, 2014):

- It's a simpler implementation than WebSockets since there is no need to worry about multiple concurrent connection from the same client, it's very well adopted by the industry, some companies or clients can have some difficulties implementing WebSockets on their side and some servers, for example proxy servers might still have some difficult dealing with WebSockets;

- It’s easier to document since the different kinds of requests follow a standard protocol, there are libraries and tools that help automating the documentation process (code first), like Swagger[20], there is no such functionality for WebSockets, this is because WebSockets is a lower level protocol and only worries about how the connection and messages are delivered, the messaging protocols, rules and methodologies of the messages are defined by the developer, for example, all the creates, deletes and updates resources and status messages must be built on top of the WebSockets implementation;
- WebSockets is typically used as a stateful protocol while RESTful as a stateless one, this makes REST easier to scale horizontally.

2.3.2.1. HTTP/2

It’s also important to note that the current limitations of RESTful communications might change in the future with HTTP 2.0 (RFC 7540), the primary objective of HTTP/2 is to enable more efficient network connections, reducing the transfer data size and enabling multiple concurrent data transfers on the same connection which is not possible with HTTP current version (HTTP/1.1) (Belshe et al., 2015).

Some of the advantages HTTP/2 will bring can be summarized has follow (Belshe et al., 2015):

- **Multiplexing:** each HTTP request/response can be associated with its own stream which are independent of each other, meaning one blocked request does not prevent another's progress;
- **Flow control and prioritization:** this ensures that multiplexing is efficiently used. Flow Control controls data transmitted and helps to ensure that only that data can be interpreted by the receiver that is transmitting. Prioritization helps in prioritizing the most important data first;
- **New interaction mode:** with HTTP/2 servers can push data to the clients, for example, data the server knows that the client will need. For the server to be able to do this, it first needs to synthesize a request to send a response to;
- **Compression:** frames that contain HTTP header fields are compressed reducing the size of transmitted data;
- **Connection Management:** connections are persistent and are not closed until determined that no further communication with a server is necessary, for example, when a user navigates away from a web page or when the server closes the connection. This allows many request and responses to use the same connection.

HTTP/2 is already being used by some major companies, like Google for example, but this will take time, so for now it still is important to consider HTTP/1.1 first.

20 https://swagger.io/

Also, it's just recently that libraries and frameworks are supporting HTTP/2 and even the ones that support it are at the state of pilot testing and advice not to use it in production, this is the case of Play Framework with the lightweight Akka HTTP server, that, since version 2.6 supports HTTP/2[21].

HTTP/2 is not in the scope of this work, but it's important to be aware and prepare for future changes.

2.3.3. Frameworks

There are many frameworks, in different programming languages, that emerge to facilitate software development in reactive paradigm: React and NodeJs for Javascript (nodejs.org); Reactor, RxJava, Reactive Extensions for .NET, Rx.rb for Ruby, the Play Framework for Java and Scala and many other initiatives.

2.3.3.1. Concurrency model

There are two types of concurrency models, thread-based on a call stack and shared memory and message-driven or event-driven concurrency. Many popular frameworks, like for example Ruby on Rails, are thread-based, this includes features like, a thread per request and concurrent access to mutable states are managed with locks and other complicated constructs (Zhu et al., 2015).

Furthermore, managing thread pools, can be difficult, if the thread pool is too large it consumes to much resources and it it's too small it can run out of threads, for example in a spontaneous increase in network traffic, also, a service latency can affect another's and so one, making thread pool optimization very hard to handle (Brikman, 2013).

To solve the previous problem, this work is developed on the principals of message-driven concurrency, meaning, in most cases, a single tread per CPU core, because of this, these threads need to be non-blocking (all I/O should be asynchronous), so that they can process other request until a response for the request is ready at which time it relays the response, for this to be possible the requests are inserted into an event queue and each event is associated with an event callback (Zhu et al., 2015).

Below are two pictures of each concurrency model for better understanding.

21
https://www.playframework.com/documentation/2.6.x/AkkaHttpServer#HTTP/2-support-(experimental)

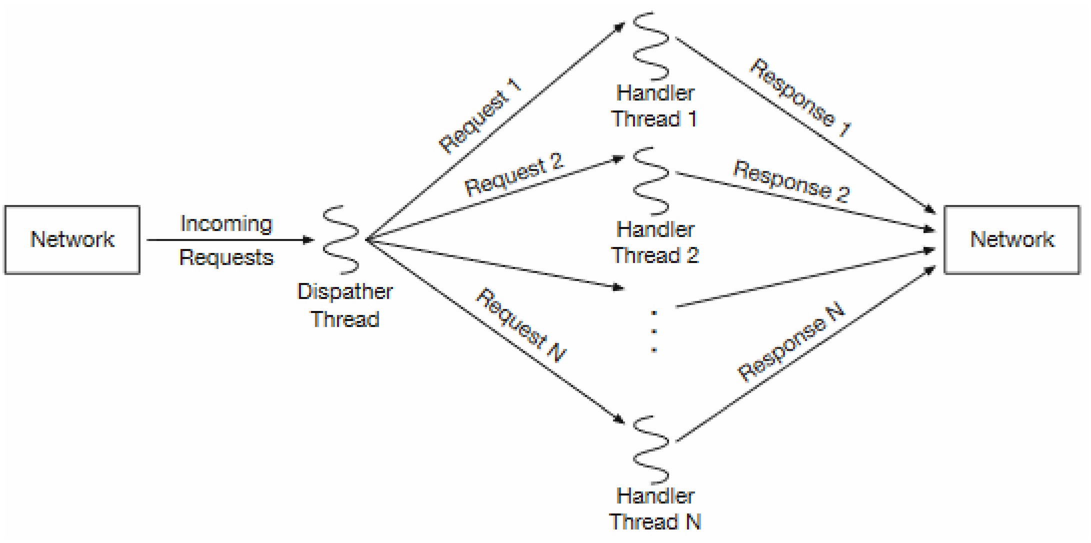


Figure 6: Thread-based execution model (Zhu et al., 2015)

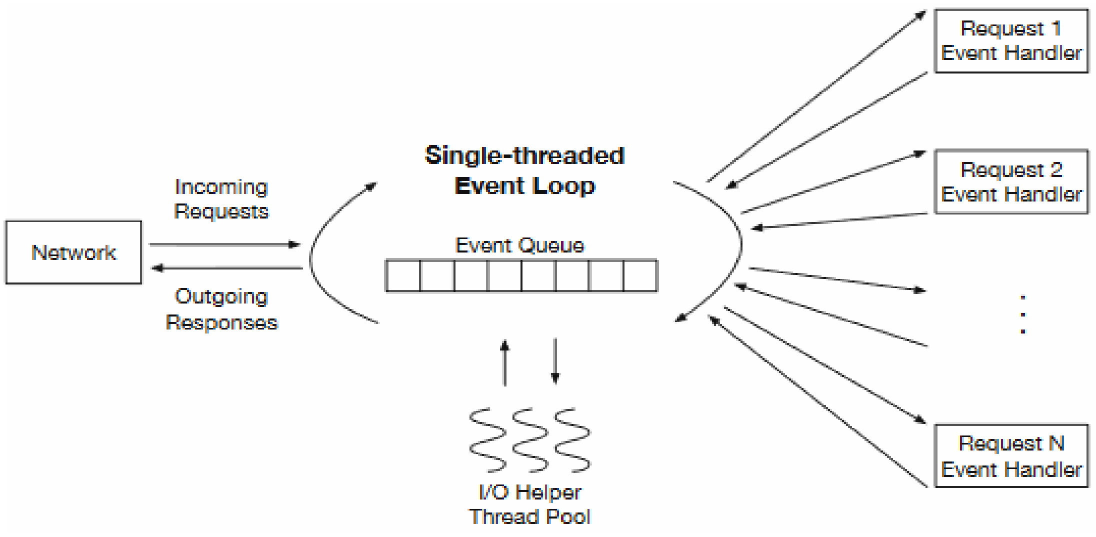


Figure 7: Event-based execution model (Zhu et al., 2015)

Asynchronous message-driven designs are not all without their problems and complexities has well, if they are built poorly, they can lead to hard to solve problems. Producing asynchronous code, in general, it's a little more difficult than producing synchronous code, and the developer should adapt and understand the differences between the two. That's why is important to rely on proven stable frameworks and ecosystems to help develop this kind of architecture, and this is where Play Framework comes in (Brikman, 2013). More detail on developing asynchronously in chapter 3.1.4.

2.3.3.2. Play Framework

The Play framework is the chosen and main framework this project uses to achieve the reactive characteristics of the platform.

The Play Framework is a full-stack web framework that was built with the reactive manifesto in mind, it's libraries and structure helps the development of more scalable and responsive web applications in the JVM space. Play currently supports the development in Scala or Java, but because it was built mainly in Scala, some parts of the system need to be coded in Scala, and overall Scala is better supported, for example, the routes file, which is the default request routing mechanism uses a Scala Domain Specific Language (DSL). It was also built on top of technologies like Akka and Netty[22] so to be fully asynchronous and work in a non-blocking I/O when needed. This also facilitates the use of parallel I/O calls to improve the use of real-time technologies like WebSockets (Brikman, 2013).

2.3.3.3. Akka and complementary technologies

Akka is an actor-based toolkit and runtime, part of the Typesafe Reactive Platform[23] for building highly concurrent, distributed, and fault tolerant actor-based applications on the JVM. Akka has a number of other incredible features for building Reactive applications, like supervisor hierarchies for resilience and distributed workers for scalability. Akka helps in separating the business logic from the complex logic involving threads, locks and non-blocking I/O, and helps the developer abstracting from the challenges of managing state and the location of services.

Akka is also a major component is complementary technologies like for example SMACK (Spark, Mesos, Akka and Kafka) which from a conceptual point of view are join together to provide even more power and flexibility in building and maintaining high demand and distributed systems. These technologies are not in the scope of this work, but it’s important to have understanding of what is the state of the art with these technologies and how they complement each other.

2.3.4. Micro-Services

Like mentioned before and one of the objectives of this work, an ESB is one possible approach to implement a services mediator, ESBs has a whole, bring a lot of complexity to the implementation and configuration of the services they intent to deliver, but it’s functionality it’s still useful. In this respect, and as a possible approach to this problem, it’s possible to divide the ESB into smaller services, thus reducing the complexity of each individually service, meaning that there can be a compromise between understanding completely one service while having doubts about another.

From a developers point-of-view, a lot of lightweight micro-services put together can do the same has a ESB SOA oriented architecture and it’s the direction where the industry is moving to. Some say that ESBs are dying, while others say that the definition of an ESB doesn’t directly affect it’s implementation architecture and that the definition of an ESB is still quite useful (Stafford and McKenzie, 2014), ergo, the technology has evolved.

22 Current version, 2.6.x, replaced Netty Server with Akka HTTP Server

23 https://www.lightbend.com/products/reactive-platform

2.3.4.1. Monolith Vs Micro-services

According to Randy Shoup, in a Q&A interview from the 2017 QCon software development conference in New York, companies main focus should be on the business and not the technology (Churchville, 2017), that being said, software has become a major tool from companies in which most companies are also software companies (Callahan, 2017).

A monolith is not a pejorative term, it's still an adequate solution in many scenarios, that being said, there are many examples were successful companies evolved from a monolith architecture to a micro-services architecture, meaning that when scaling was a major concern, the monolith architecture was not a good fit (Churchville, 2017).

> "as I add more people to my team, everybody is stepping on each other's toes in the monolith, and they're just slowing everybody down" (Randy Shoup, 2017)

Twitter is a good example where it started as a monolith and when it started having problems with high load and scaling, it turns to micro-services has a solution, breaking the monolith architecture into smaller pieces (Churchville, 2017).

2.3.4.2. Isolation and Single Responsibility Principle

When designing micro-services, isolation is one of the most important aspects of it, affecting not only the technical components but also the people, in their tasks, responsibilities and their place in the organization (Bonér, 2017).

> "Any organization that designs a system (defined broadly) will produce a design whose structure is a copy of the organization's communication structure." (Melvyn Conway 1967)

The Unix philosophy has always been to write programs that do one thing but do that thing really well, programs should work well together[24] (Doug McIloy), this later came to be known in object-oriented programming has the Single Responsibility Principle (SRP)[25].

It also makes it easier to adopt continuous delivery and scale of the services since the separation of responsibilities facilitates the roll out and revert of changes incrementally. Planing changes to a monolith, because of the tight coupling between components, requires a lot more planing and testing, if something goes wrong with one components it can easily affect another component (Bonér, 2017).

2.3.4.3. Scalable Persistence

The problem: Persistence could be a major bottleneck in the platform, traditional database persistence techniques do not scale well and are typically blocking IO, SQL databases and CRUD operations do an update-in-place when updating data which means that the data is

24 http://homepage.cs.uri.edu/~thenry/resources/unix_art/ch01s06.html
25 http://www.labri.fr/perso/clement/enseignements/ao/SRP.pdf

replaced with the new one and during this time the same data is locked. So, a strategy must exist to better deal with high data demands.

Possible solutions: An alternative to this is with the pattern Event Sourcing[26] which with every triggered state changes the corresponding instruction is saved in an event log never locking previous data, this log represents all the data at any specific time (Fowler 2005). This has a big drawback which is complexity, packaging every change in an event is not something that comes natural to all developers (see footnote 26).

Also, the CQRS pattern which stands for Command Query Responsibility Segregation, coined by Greg Young, is used to alleviate the blocking IO. CQRS introduces a change in the conceptual model, in that the reads and writes are separated, the reads are referred to as Query, and the updates as Command. This is not without some disadvantages, implementing CQRS can bring complexity to the platform, which goes against one of its primary objectives of staying simple, because of this, CQRS is used only if it's clear that will benefit the platform in performance (Fowler, 2011).

A major benefit of using CQRS, is that the scaling can be done separately, if the reading part needs more resources that it can be up-scaled without worrying about the writing components (Fowler, 2011). This is specially useful when the reads and writes are very disproportionately, which is the case for this platform in that it will be dealing with more writes than reads.

Combining Event Sourcing with CQRS could provide a viable solution the to the blocking problems but it has a drawback that is dealing with the full complexity of the system in the very beginning of the development, it's also important to understand other implications like that consistency between the reads and writes will not always exists at all time, the system needs time to propagate the new writes to the reads portion of the system (Bonér, 2017).

2.3.4.4. Monitoring

Micro-services aren't without its disadvantages, and, in a distributed system, tracing a problem can be a difficult process especially in a reactive system. The easiest way to deal with this is using logging across the entire platform, which can provide a detailed view of the systems, but this alone may not be optimal. So, to help with this, the Akka framework already provides mechanisms for improving monitoring capabilities, which are (Lightbend, 2017a):

- **Akka supervisors:** Supervisors are the actors fathers. An actor that creates another actor can become its supervisor and this allows the supervisor to monitor its children state and even recover from failure if something wrong happens to its children, like resume, restart, stop the actor or escalate the failure to the Akka top level actor supervisor (also called "the one who walks the bubbles of space-time") (see figure 8). Reacting to actor failures can also be done with a delay with the BackoffSupervisor pattern (see code 1), this is useful, for example, to give time for the source of the

26 https://martinfowler.com/eaaDev/EventSourcing.html

problem to be fixed before restarting the actor. Each actor can only have one supervisor, but the supervisor can have any number of children, this encourages sound design decisions and simplifies actor hierarchy. The supervisor is just like any other actor but it's best to keep the supervisors as simple as possible, because if a supervisor fails, all it's children fail has well. Another thing to keep in mind if that there are two ways to supervise other actors, one-for-one strategy or all-for-one strategy, the former meaning that the supervisor acts only on the children that throw the exception, and the latter meaning that the supervisor acts on all children despite only one of its children had a problem (Lightbend 2017c).

```
val supervisor = BackoffSupervisor.props(
  Backoff.onStop(
    routerActorProps,
    childName = "router",
    minBackoff = 3.seconds,
    maxBackoff = 30.seconds,
    randomFactor = 0.2 // adds 20% "noise" to vary the intervals
  ).withAutoReset(10.seconds)
    .withSupervisorStrategy(
      OneForOneStrategy(maxNrOfRetries = 10, withinTimeRange = 1 minute) {
        case _: ActorInitializationException => SupervisorStrategy.Stop
        case _: ActorKilledException => SupervisorStrategy.Stop
        case _: DeathPactException => SupervisorStrategy.Stop
        case _ => SupervisorStrategy.Restart
      }
    )
)
```

Code 1 – BackoffSupervisor strategy (Lightbend, 2017c)

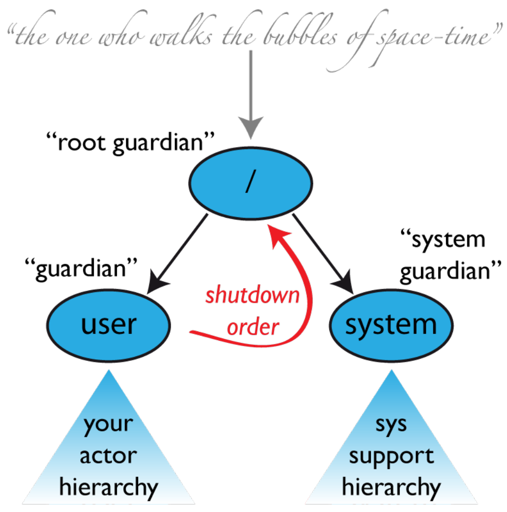


Figure 8: Akka supervisor hierarchy (Lightbend, 2017c)

- **Akka monitoring:** Lightbend also provides Intelligent monitoring features in it's enterprise suit, which is a commercial feature but also a comprehensive and advance tool (Lightbend, 2017b) (Lightbend, 2017c);

- **Akka telemetry:** Telemetry is part of the monitoring features and is used to capture events and metrics from the actor system. Telemetry is free to use in development but not free to use in production, for this, it requires a subscription from Lightbend. Telemetry events are a set of metrics that trigger when they reach a specific threshold and they can be configured, for example, when the number of running actors exceed a specified amount. Some events and metrics that can be captured by the telemetry are: Actor local and remote (running actors, mailbox size, stash size, processed messages, sent messages, failures, dead letters, etc..), cluster (domain events, member events, etc.) and dispatcher (ForkJoinPool, ThreadPool metrics, etc..). Akka telemetry also allows the tracing across a distributed system in separated networks and JVMs (Lightbend, 2017b).

In a large distributed system, the number of active actors can easily go beyond 100 or 1000, the lifespan of an actor can vary greatly and the actor system can even spread across a cluster of networked nodes, so, monitoring is very important and should be taken seriously or problems could escalate very quickly and control of the system lost.

2.3.5. Security

Security in general is important, in this project however, is one of the most important factors because the stakeholders involved, want to protect/keep their data confidentially from any other element besides the one with whom they are communicating. With these kind of sensitive information, the damages can go from illegal manipulation of information to life threat damage (Kang et al., 2016).

There is more than on layer in security, but to keep things simple, it can be divided in endpoint security (or end-to-end) and data privacy.

End-to-End security means that security is implemented exclusively on the endpoints of a connection, meaning the client or the server, client-server or client-client, and usually contains the following components (H. Behringer, 2009):

- Identity: User authentication and authorization;

- Protocols: For example, Transport Layer Security (TLS);

- Algorithms: For example, Advanced Encryption Standard (AES);

- Secure implementation: The software implementation must be stable and free of bugs, otherwise these can be exploited for security workarounds;

- Secure Operation: Any operator of the system should understand security principles, like detecting invalid certificates.

Endpoint authentication and authorization is provided by all ESBs referred in this document (see chapter 2.7.5.4), so it's a common thing, data privacy however is not, this is because privacy is a very complicated thing and usually the company providing the services has access to the data they are managing, but one of this project requirements is the protecting of data from endpoint to endpoint, this means that the services relaying the messages cannot have access to the data unencrypted content. Data privacy, sometimes referred to has end-to-end encryption, goes beyond endpoint security.

2.3.5.1. Rate Limiting

Rate limiting is the ability to restrict requests based on the number of requests per period of time, in its definition it's not a security feature, but it secures services with the ability to prevent abuse from malice and from unintended consequences, like for example, from a client with a bad developed script that's making too much requests (Raghavan et al., 2007). Its limits should vary based on the available resources and possible the importance of the request, critical requests should have priority over lower ones (Rao, 2011). They help managing high volume traffic and their limits, by placing caps on the traffic. Configuration should be configurable from outside the code to allow for ad hoc changes and/or dynamic changes based on the current state of the system.

There can be several types of rate limiters but they all accomplish basically the same result which is an intentional denial of service. Some different types of rate limiters can be (Raghavan et al., 2007):

- User/IP based rate limiters: rate limiters can be based on the source IP address or in the user account making the requests;

- Request rate limiter: this are basically the previous description and are the most basic form of a rate limiter, meaning they block requests when these surpass the allowed amount in a period of time;

- Concurrent request rate limiters: instead of limiting request based on a period of time, these rate limiters limit request based on the number of active connection from the same source, IP or user. These are useful in controlling more resource intensive consumption requests, i.e., when even a smaller number of requests use too much resources of the system.

A possible and popular approach in implementing rate limiting is to use the token bucket[27] algorithm, this approach requires that every request retrieves a token from a bucket, if there aren't any tokens available, then the request is denied, tokens are added to the bucket in

27 Possible token bucket alternatives: leaky bucket; hierarchical token bucket

defined intervals of time dictating how much requests can be made in a period of time (Raghavan et al., 2007).

Tokens should be added to the bucket in a roll-out / sliding window fashion, meaning that if the intended limit is 20 requests per minute, tokens should be added through-out that minute and not add 20 tokens per minute. Following the previous example, too keep an evenly roll-out of tokens throughout 1 minute, the token **(T)**, or requests, rate at which to add to the bucket**,** can be calculated based on the defined time period (in this case 60000ms) **(M)** and the number of requests **R** that should be allowed for that same time (see next formula).

$$T=(M/R) \tag{1}$$

For example, to keep 20 requests per minute 60000/20=3000, which means that a token should be added every 3 seconds. If the tokens are already at max capacity, 20, then no token is added. This also means that it's possible to make more than 20 requests per minute, but that's intentional, because the idea is to prevent abuse and not service, the 20 requests per minute is a guide line, meaning that if requests come two fast it will it that limit and service will be denied, but if the requests arrive, for example, every 2 seconds, then the request will be allowed to surpass the 20 limit requests per minute, if for example, the service being requested is very CPU or memory intensive, then the limit could be increased and also the rate at which the tokens are added. Important to note that this formula is useful in setting an initial configuration for an intended limit, but it's also necessary to validate and test the limits in a case by case basis, and if needed the limits can be changed accordingly based on the results of the tests.

- Direct memory access Vs Redis

The fastest way to access a data store is with direct, local, memory access, but, with the increase in traffic and complexity of the whole system it's more optimal, to use a more focused purpose in-memory key-value data structure store, like Redis[28] [29], which is one of the most popular key-value store today and one of its typical use case is rate limiting (Amazon AWS, 2017). Local memory access can degrade in performance when the data grows bigger, depending on the local resources and on garbage collection configurations, Redis is design to deal with these situations specifically, meaning that it can work with separate processes, run on different nodes and implement high-availability.

Amazon AWS also provides ElasticSearch[30] which can take Redis and turn it into a distributed, fast and scalable solution, prices start at 0.017€ per hour for 1 instance of a data-store with

28 https://redis.io/
29 Possible Redis alternatives: MongoDb and Memcached
30 https://aws.amazon.com/elasticache/

0.55Gb for a low to moderate performance, from there the price goes up depending on number of instances, memory space and performance requirements.

- Rate limiting concerns

Despite its advantages, there are some concerns that need to be addressed, if something goes wrong, when implementing rate limiters: there should exist functionality to deactivate them, they should report back to the client with clear messages, so that the client knows why the connection are being dropped, if they use an external data-store, the requests should not be compromised if the connection to the data-store is lost and they may need constant tuning depending on the request and/or traffic.

2.3.5.2. Load shedding

Load shedders, or load aware shedding, differs slightly from a rate limiter in that it makes it's decisions based on the whole system rather than just on a component or access source, their purpose is to avoid excessive consumption of available resources that could compromise the entire system, instead, load shedders can be used to drop some traffic in favor of other, more important, traffic. They are most useful in complex systems and on situations of emergency where critical/core systems must be kept operational while the rest of the system might be offline. Their implementation, however, are non-trivial and more complex than implementing rate limiters (Rivetti et al., 2016).

2.3.5.3. IP Black Lists and White Lists

While lists are typically used for two reasons, for just allowing traffic from its IPs or, for not applying restrictions, like rate limiting, to its IPs. This is useful for defining limits and restriction bypass, being this per-determined or on-demand configuration.

Black lists serve only one purpose which is blocking access, typically this is used to block access to problematic IP addresses.

2.4. Deployment

The infrastructure itself is not a constraint for the platform, but it can make its deployment a lot easier if its business model is in sync with the platform, meaning that, it should be easy to adapt to a micro-services architecture.

Another important issue is with storage, since the platform deals with sensitive user data, there are laws that mandate that, some data, stays inside the country.

The following data is all gathered from the official websites with the exception of Claranet that only provides estimates following a commercial contact.

2.4.1. Infrastructure Providers

Several providers were analysed in terms of location, business model, reliability, cost and ease of use, among those, some proven to be more compatible with the platform than others.

Table 2: Cloud providers list

Provider	Description
DigitalOcean (digitalocean.com)	DigitalOcean is a cloud provider which focus on providing simple and cost-effective services. It allows control of all tasks online.
Azure (azure.microsoft.com/pt-pt)	Microsoft cloud provider, it's main focus is Microsoft technologies. It allows control of all tasks online.
AWS (aws.amazon.com)	Amazons AWS provider, it's one of the biggest providers for Large Enterprises but it's also more complicated to integrate with since the number of options and configurations are many. It allows control of all tasks online.
Claranet (claranet.pt)	Claranet is a Portuguese provider, which also means that has datacenter in Portugal, in fact, it's the only on from the list with local datacenters. It's also planing to be an Azure datacenter provider[31]. It does not provide online control panels, it has some partial solutions, but mainly it counts with client support to provide client configurations requirements.

2.4.1.1. Location

In terms of location most providers are international with the exception of Claranet. Location is not a major concern in most cases, but it's important to be aware of the data and the laws that abide it. If in some case the law dictates that the data must be keep inside the country them it could present a limitation in choosing a provider.

2.4.1.2. Business Model

The business model of all providers is basically the same, they charge based on the resources used, but there are some differences. AWS and Azure give high priority to the data and it's consumption, meaning that the larger the data and/or traffic the bigger the cost, the idea is the same, to provide smaller companies a more cost effective entry point, this comes with a disadvantages which is more complex options in regards to the way they estimate the cost by the kind of service provided.

DigitalOcean has a simpler business model, mostly all costs are determined by the virtual machines used, CPU, memory and storage, traffic has some limitations but it is usually not a

31 https://www.computerworld.com.pt/2017/05/30/claranet-portugal-prepara-datacenter-azure/

concern since the higher the resources the higher the available traffic, contrary to AWS and Azure that provide containers to service, DigitalOcean only provides virtual machines called Droplets that are basically Linux machines, windows is not supported, the drawback is that it's the client responsibility to keep services online but has the advantage of being a much easier business model which also means that planing for expenses in the future is easier.

Claranet is a little different, they don't have an automated business model has the other providers, they provide personalized solutions to enterprises, this requires a contact with they sales department, technical meetings with their IT leaders and technicians and after that they construct a business proposition based on the client's provided requirements and size. Support is a little more hands-on than the other providers which might be a requirement in some cases, but this also comes at a cost.

2.4.1.3. Reliability

There is no guaranties on availability by any provider, but all of them are in business for more than five years and with no loss of service reports. AWS and Azure have the bigger portion of market from which one could claim that they represent a higher reliability than the others providers, but there is no data publicly available that undermine the other providers in this respect.

2.4.1.4. Cost

To better lodge micro-services, a scenario with more less powerful virtual machines is preferred to less more powerful virtual machines, so it's important to focus on the cost of the cheapest virtual machine. As previous stated, all values of the table below were extracted directly from the official providers website with the exception of Claranet.

Table 3: Price comparison between infrastructure providers

Provider	Low power virtual machine	Monitoring, Backup, Security, Additional Storage and Load Balancing	VM Cost p/month
DigitalOcean (digitalocean.com)	RAM: 512Mb CPU: 1 Storage: 20Gb SSD Transfer: 1Tb	Monitoring: 0€ Firewalls: 0€ Backups: 20% of the VM cost, so in this scenario 1€ per VM. Additional Storage: 10€ for 100Gb Load Balancer: 20€ p/month	5€
Azure (azure.microsoft.com/pt-pt)	RAM: 0,75 Gb CPU: 1 Storage: 20Gb Transfer: *	Monitoring: 0.009€ per 1 000 API calls and then it charges for email reporting 1.687 € per 100 000 emails Firewalls: 0€ Backup: ~4.2€ for 50Gb or less. Additional Storage starts at 678 Gb for ~ 233,40 €	~11,30€

		Load Balancer: *	
AWS (aws.amazon.com)	RAM: 0.5 Gb CPU: 1 Storage: 20Gb Transfer Out: 1Tb	Monitoring: 0€ for basic monitoring Firewalls: 5€ p/month p/firewall plus 1€ p/rule plus 0.60€ p/million web requests. Backups: 0.0245 p/Gb (0.5€ p/20Gb) Additional Storage: 0.0245 p/Gb (0.5€ p/20Gb) Load Balancer: 21.6€ p/month plus 0.008€ p/Gb of data processed	6.63€ + 90€ (1Tb transfer out)
Claranet (claranet.pt)	RAM: 1Gb CPU: 1 Storage: 20Gb Transfer: Unlimited	Monitoring: * Firewalls: 0 € Backups: * Additional Storage: Additional storage is sold by Tb => 300€ = 1Tb Load Balancer: 250	~30 €

* These values were not possible to obtain. Claranet does not charge this has separate solutions, and the others vary too much based on the type of service being used.

Azure and AWS costs vary based on the location of the data and servers, the values of the table above are for Europe. All values are true for the time being (July 2017) and may change in the future. Azure and AWS have very complicate configurations, the setup in the table above was made simple so that a comparison between the different providers was possible and just. Claranet is an enterprise focus datacenter and the price varies based on how much you buy, this means that a VM can become cheaper if you have many, but this is always subject to negotiation.

There are a lot more datacenter providers, these few were chosen based on what they offer for this project.

2.4.1.5. Ease of use

DigitalOcean is the easiest to use, since it has less options than the others, it's interface it's very straight forward and practical, even for someone that never used it and it's very easy to start using its services and create a test environment. It has no free trial period, but it's very easy to find a promotional code that lets one use this platform for a month without having to pay for it.

Azure and AWS have trial periods, but they required a little more effort into knowing the platform before starting to use it, it's not has straight forward has DigitalOcean, mainly because they offer more services, and so, it requires a previous study of the platform before even starting testing it.

Claranet doesn't have a platform, i.e., they don't provide a software interface to their datacenter, this makes Claranet the more difficult to start with. However, if the client already knows very well what it wants, he just needs to contact the sales department and ask for an estimate, Claranet will charge for it, but will take care of the initial setup.

2.4.1.6. Conclusion

In the end, DigitalOcean was the chosen one for the VMs, services and persistence storage, because it is the more compatible with the platform. It's super easy to use, the web based GUI makes managing the virtual machines and its services a good and professional experience, it doesn't have datacenters in Portugal but has in Europe and it's possible to choose in which datacenter the VMs are created and all VMs have public IPs, firewall and monitoring services for just 5€ a month.

For the backups off-site storage Amazon's AWS was chosen, in the remote event of a DigitalOcean catastrophe. Storing data in AWS is cheaper than DigitalOcean but it's still necessary to consider transfer out costs since AWS has higher costs for download traffic the for upload traffic, but since backups are only needed in case of an emergency or loss of data, it can be assumed that these prices should not escalate.

Important to note, however, that it's still possible to use Docker[32] with a virtual machine solution provider, like DigitalOcean, through Docker VM Drivers, Docker is a container and not a VM (Coleman, 2016), but that's not inside this project scope, mainly because Docker is a container solutions and does not really solve the problems of this work and it would just had complexity to the project, has the project grows Docker solutions might be considered.

2.5. Quality

Quality in healthcare can be divided in 3 main domains (Lupo, 2015):

1. Management quality: includes procedures and methods and should apply effective resource management, to satisfy stakeholder needs and expectations;
2. Professional quality: includes skills and equipment and is affected by personal perspective of the same;
3. Stakeholder perceived quality: includes the stakeholder perception of service provided quality.

The last one is considered the most important one because it tells us the satisfaction level of our stakeholders.

32 www.docker.com

To divide and classify stakeholder satisfaction, Tony Lupo's approach was used and adapted to this project, which resulted in the following categories, seen in the following figure (Lupo, 2015):

- Healthcare staff: How can this project improve staff's work?
- Responsiveness: Are the services provided by this project responsive enough?
- Relationships: Is there trust in the data and the in the privacy of the same by the stakeholders? Can useful information about work related variables be easily accessed, like for example, product information?
- Support Services: Do the stakeholders have access to convenient support infrastructures whenever they need? Is the support useful?
- Accessibility: Is the services provided by the project always available? Do they provide important information about work procedures/processes? Is the information easy to access?
- Tangibles: Is the data produced by the services useful and trustworthy?

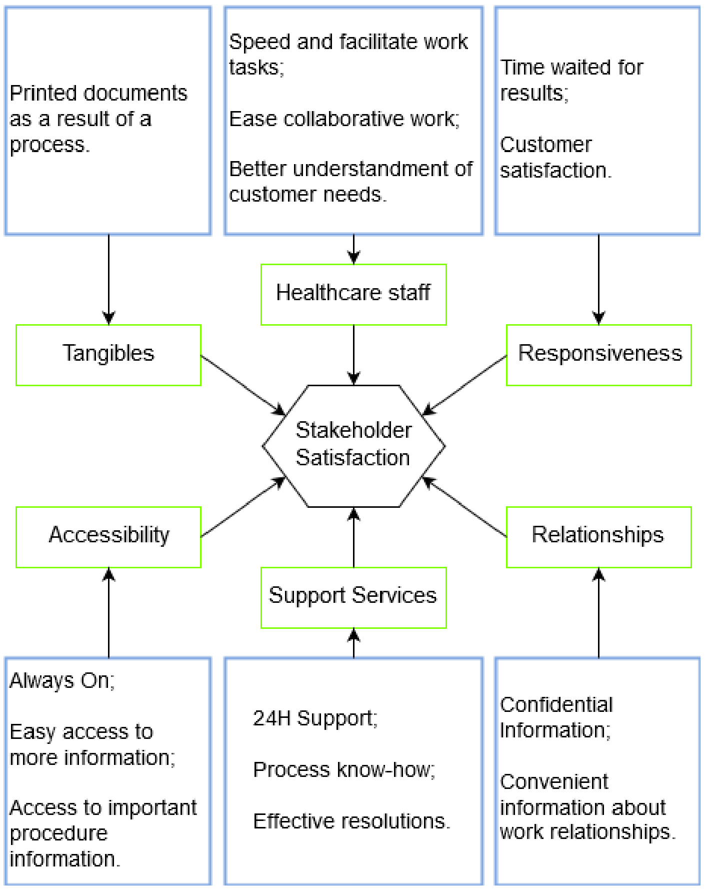

Printed documents as a result of a process.
Speed and facilitate work tasks;
Ease collaborative work;
Better understandment of customer needs.
Time waited for results;
Customer satisfaction.
Healthcare staff
Tangibles
Responsiveness
Stakeholder Satisfaction
Accessibility
Relationships
Support Services
Always On;
Easy access to more information;
Access to important procedure information.
24H Support;
Process know-how;
Effective resolutions.
Confidential Information;
Convenient information about work relationships.

Figure 9: Quality categories (Lupo, 2015)

2.5.1. Stakeholders

To represent the stakeholders an Onion diagram was used (see figure 10), this diagram displays the stakeholders in a layered form, being the inside of the Onion the most closest to this project and the outer layers the farthest or external relationships to this project. It also shows what stakeholders are connected to one another (Alexander, 2003). It's a simpler, more high-level approach, to the value networks in chapter 2.7.2.

The first, inner, layer is usually referred to as the Product or Solution layer, the second layer can be seen has the business system that represent the stakeholders that interact directly which the project/solution, the third has the business itself, usually the managers, sales, etc. that link the business system with the external entities, and the fourth layer as the Environment which represents the stakeholders that are external to the organization. There can be a fifth layer if needed that represents the stakeholders that don't relate/map with any other stakeholder. The arrows represent a connection to another stakeholder (Alexander, 2003).

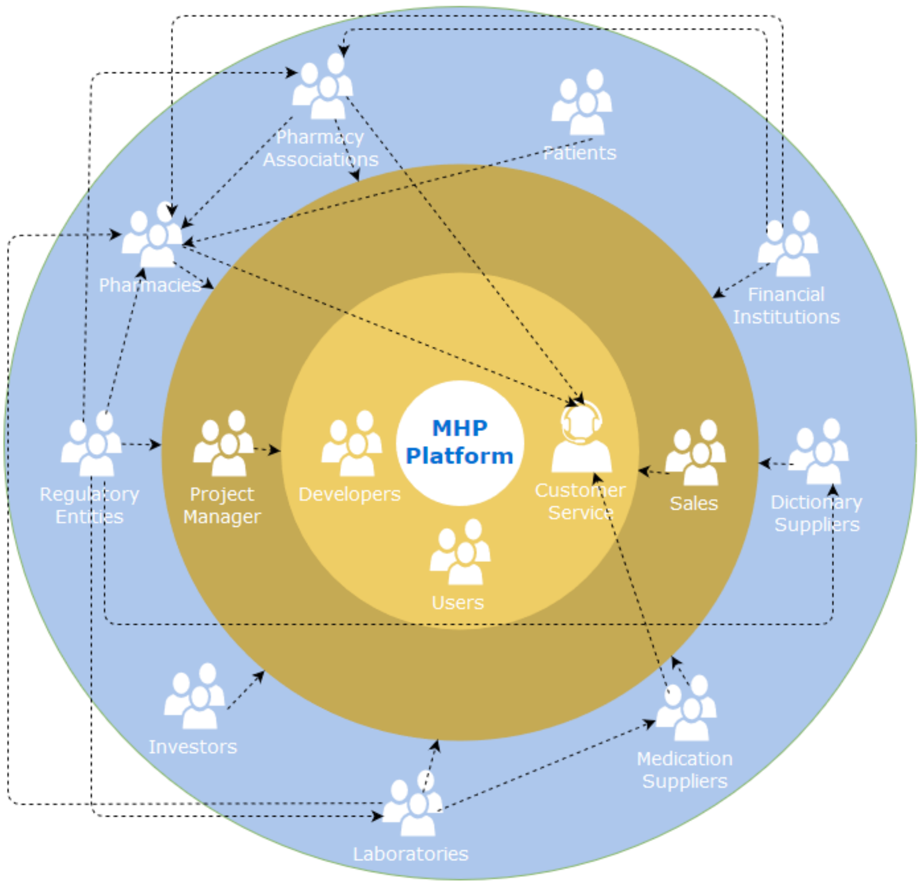


Figure 10: Stakeholder relationships

2.6. User Data Protection

User data protection, not only is an important feature to have, reassuring the stakeholders that their data is safe, but it's also mandatory, in some cases, by legislation.

A new directive, RGPD (*Regulamento Geral de Proteção de Dados*), will enter in effect in May 25, 2018, which replaces the current one, and brings several changes that have a significant impact on the lives of the organizations, depending of course of their nature. There are 10 main points that need to be addresses, which are has follow (CNPD, 2017):

- Information provided to the data owners or sources: At the moment there are no specifics, but more information and more concise information will need to be provided to the users, with special care for children, this means, for example, that privacy policies and forms to retrieve data will need to be reviewed and more likely than not, changed;

- Guaranty the data owner's rights: for example, users that request access to their data should receive an answer in a defined period of time, changes or deletion of the data will also be regulated. Because this affects the rights of the citizens directly, there has been several changes in this area and organizations should prepare for its adaptation and implementation. All process must be well documented;

- Data owners consent: The method and circumstances that the owner's consent is acquired must be validated and proven, if this is not the case then a new consent is required to keep or use the data. There are also special cases when dealing with children;

- Sensitive data: Data must be validated in order to define what can be considered sensitive data and what can be subject to special conditions, for example, biometric data is now part of the sensitive category that this directive extended;

- Documentation and registry of data management activities: All activities related with data treatment must be documented, the organization must be able to prove that is respecting all obligations imposed by the RGPD. This measure is especially important, because it allows the validation of all that is being done and also what is needed to fix or adapt. If this measure is not accomplished, the organization might need to start from the beginning in gathering the data;

- Outsourcing / Subcontracting: outsourcing contracts must be subject to the same rules, it's the responsibility of the subcontracted to verify that it has has all necessary authorizations from the entity responsible for the data. All authorizations must be squired before May 2018;

- Entity in charge of the date protection: The must be someone designated has the responsible for the data, i.e. responsible for its protection, this person should be able to report directly to the highest level of the organization;
- Technical and organizational measures: The organization's policies and practices must be reviewed to assure that all RGPD regulation are met. In this evaluation the nature, context and purpose of the data must be identified, has well has the danger it represents for the safety and liberty of the citizens. This measure also allows the verification that the data has all the necessary treatments and that is safe from deletion, corruption and confidentiality loss;
- Data protection since its conception and impact validation: In order to decide and implement the best possible measures, data treatments must be rigorously validated in an early stage, i.e., before data gathering;
- Security violation notifications: Internal procedures must be adopted in order to detect and report any kind of data violation. Not all violation must be reported to the CNPD, but all should be documented.

2.7. Value Added Analysis

This section provides an overview of the added value of this work by describing its key focus points.

2.7.1. Value Proposition

This work creates a unified communication platform that interconnects all healthcare entities promoting, unifying and facilitating the flow of information, thus improving the quality of the data.

2.7.2. Value Network

The following diagrams shows the exchange of roles and monetary value that intends to show the specific value this project can generate. In general, it shows (Allee, 2012):

- How the work actually gets delivered;
- The kind of generated value;
- How efficiently this network converts resources into value;
- And in what points of the network could generate problems and/or inefficiently.

For better clarity the value network was divided in two diagrams.

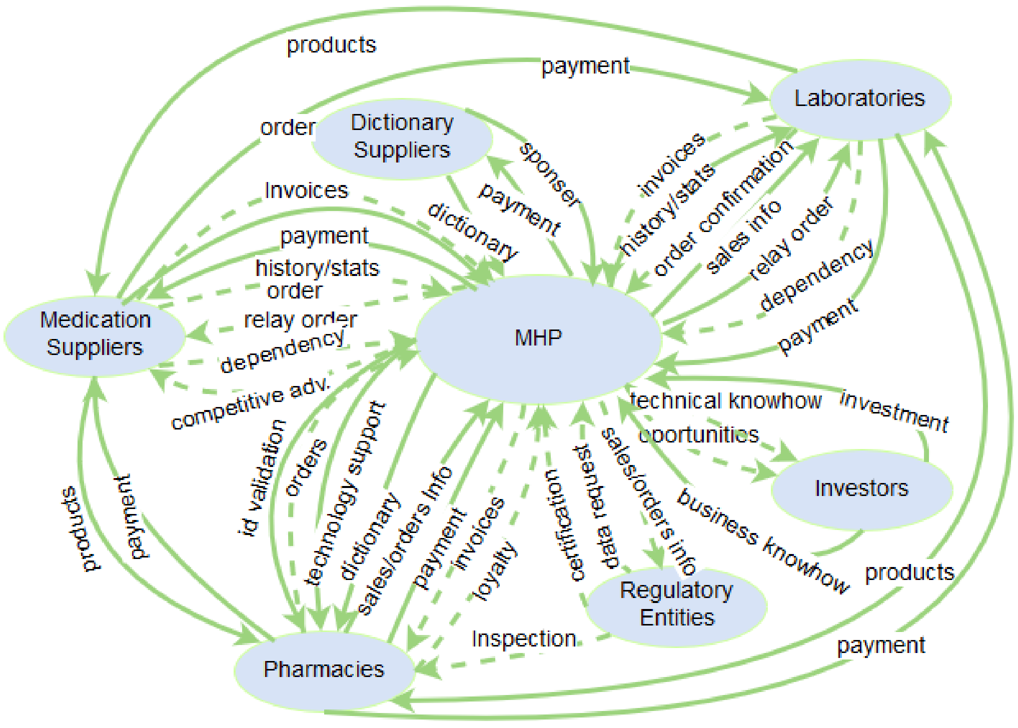


Figure 11: Value Network 1

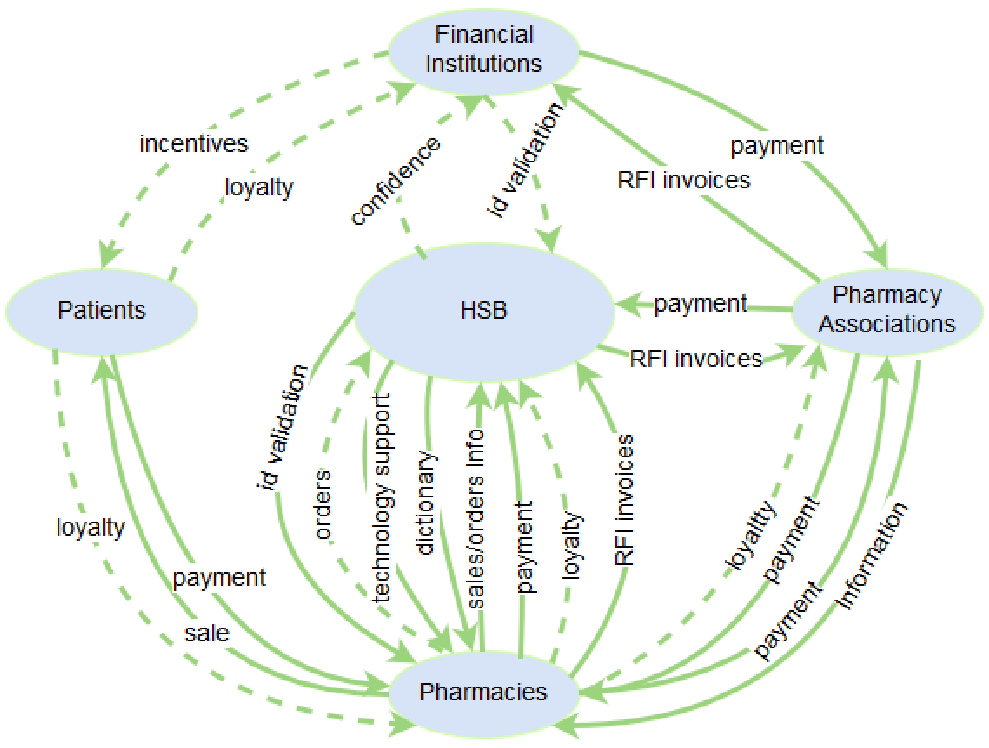


Figure 12: Value Network 2

Table 4: Value Network Roles

Role	**Description**
MHP Services	Represents this project provided services.
Pharmacies	Pharmacies that interact with the MHP.
Patients	Pharmacy clients.
Medication Suppliers	Pharmacy medication suppliers.
Dictionary Suppliers	Entities that supply useful and validated information, usually in the form of a database. These entities usually only work with IT related companies.
Laboratories	Medication laboratories. They can supply medication suppliers and/or pharmacies directly.
Regulatory Entities	Entities that enforce legislated rules and laws. These entities collect information regularly or on demand.
Investors	Represents the investors of this project.
Pharmacy Associations	There are two pharmacies associations ANF and AFP, both provide basically the same services in return to a monthly fee from the pharmacies. The main differences between the two is that ANF also provides financial services which are charged separately.
Financial Institutions	These institutions provide financial help to patients, each has different rules and conditions. In practice the patients that have access to this help, do not pay the entire price of the medication they get from the pharmacy, these in return must keep track of this and they charge the corresponding institution at the end of each month. There are public institutions like SNS and private institutions like EDP or CGD. The billing must go through the pharmacy association (ANF or AFP) which acts has a financial mediator between these two entities.

Table 5: Value Network Tangible Deliverables

Roles	Deliverable	Description
All	Payment	Payment for services provided.
Pharmacies / MHP	Id validation	The MHP provides patients id’s validations to pharmacies. These services add a guaranty to the pharmacy that payment from the financial institutions will not be rejected.
	Technology support	MHP services include technical support for all provided services.
	Dictionary	MHP provides important and always up to date information, vital for the pharmacies, these include, product information, prices, details, etc., legal information, medication alerts, etc. The information is updated on a daily basis.
	Sales / Orders Info	Pharmacies provide orders and sales information to the MHP, this information is then relayed to laboratories or associations. Since this information is sensitive to user data protection laws, only data that is specified in contract between these entities is permitted to be exchanged.
	RFI Invoices	Communication of Invoices to the Responsible Financial Institutions.
Pharmacies / Patients	Sale	Sales of products and services to patients.
Pharmacies / Pharmacy Association	Information	Information is provided has a service to help the pharmacies in any aspect related to their business, like for example, legal information which is always changing.
Pharmacy Associations / MHP	RFI Invoices	Communication of Invoices to the Responsible Financial Institutions.
Pharmacy Associations /	RFI Invoices	Communication of Invoices to the Responsible

Financial Institutions		Financial Institutions.
Pharmacy / Laboratories	Products	Laboratories can sell medication directly to pharmacies.
Pharmacy / Medication Suppliers	Products	Sale of products to pharmacies. This include emergency deliveries which are delivered on the same day.
Laboratories / Medication Suppliers	Products	Laboratories sell products to suppliers.
Investors / MHP	Investment	Investment in this projects of interested parties.
	Business know-how	Investors provide business know-how on how to maximize this work success commercially.
Dictionary Suppliers / MHP	Sponsor	Information gathering entities sponsor the project with the interest of making some searches favour them.
	Dictionary	Provide products and related information updates.
Medication Suppliers / MHP	History / statistics	MHP provides history and statistics on orders.
Laboratories / MHP	History / statistics	MHP provides history and statistics on orders.
	Sales Info	MHP provides sales information on specific laboratory products.
	Relay Order	MHP relays the pharmacy orders to the laboratories. Many of them don't have the technology in place to support typical pharmacy order protocols and value this contribution.

Table 6: Value Network intangible deliverables

Roles	Deliverable	Description
All	Loyalty	Loyalty is gained when the services provided are perceived has good or better.
Financial Institutions / MHP	Confidence	Confidence in that delivered of invoices and its accuracy is guaranteed.
	Id Validation	Provide validation of associate cards.
Financial Institutions / Patients	Incentives	Money incentive is giving to guaranty loyalty.
	Loyalty	Loyalty is gained through the money incentives.
Pharmacies / MHP	Orders	Pharmacy post orders directly to MHP that is then relayed to the Suppliers.
	Invoices	Order conference in the pharmacy is substantially improved because they have access the invoices electronically.
Regulatory Entities / Pharmacies	Inspections	Regular non-schedule inspections to pharmacies.
Investors	Opportunities	This project creates business opportunities to the Investors.
	Technical Know-How	Investors get access to technical know-how.
Laboratories / MHP	Dependency	A dependency is created on the MHP due to the lack of technology in the laboratory side.
	Order Confirmation	Confirmation of order delivery.
	Invoices	Invoices are send to the MHP to be delivered in the pharmacy.
	Relay Order	Pharmacy orders are relayed to the suppliers.

	Dependency	The more the MHP services are used the more dependency is created.
Medication Suppliers / MHP	Competitive Advantage	Services provided by the MHP are intended to be better with more information than its predecessors, giving the suppliers are competitive advantage over the suppliers that still rely on the old one. It's client, the pharmacy, prefers to use the new services because they have more control on their orders and access to more information, like out of stock products.
	Invoices	Invoices are send to the MHP to be delivered in the pharmacy.

2.7.3. Perceived value

Has already mentioned in chapter 1.5.2, perceived quality by the stakeholder can be the most important metric when measuring the quality of a product and/or service, following, bellow, is a list of value based drivers that can have a positive or a negative impact on the stakeholders perception.

Table 7: Value based drivers

Domain/ Scope	Product	Service	Relationship
Benefit	Easy to interact solutions Product quality	Responsiveness Reliability Technical competence	Image Trust Solidarity
Sacrifice	Product customization Price	Flexibility Price	Time / Effort / Energy Conflict

In the previous table, the value drivers are placed in its corresponding section, depending on what they influence and if they have a positive or negative effect. It's important to note, that these drivers can be viewed differently base on what phase of the product's life-cycle the customer is interacting.

- **Product**

  The product relates to the actual developed software has the following characteristics:

  - **Easy to interact solutions:** Since REST is very well integrated into most frameworks, almost all services provided have a REST interface/adapter with the exception of a few stateful ones that work better with WebSockets. Basically the protocol is chosen based on whats best for the service provided and not for what's easier to implement. There is also technical documentation of all services on how to implement them and examples to better help solving problems. This will ensure that consumers that have less technical knowledge have an easier interaction with the services provided;

  - **Product quality:** All services are developed and tested following the best practices in software engineering, key partners will use the service until proven it's usefulness, stability and performance has acceptable between predetermine parameters before it gets deployed to production, thus ensuring the confidence of the clients in the services provided. The services are also monitored constantly to assure consistency and improvements over the life cycle of the same services;

  - **Product customization:** In this work customization is considered second to the correctness operation of each provided product/service, meaning that customization capabilities may be, in some cases, hinder in favor of stability, ease of use and maintenance or development;

  - **Price:** Price is proportionally to product and services usages, meaning the more the usage the more the revenue, also, clients can better control its resources to better fits its needs. However, the development and maintenance effort dictates the price of the services, this project favors quality over cost.

- **Service**

  The service relates to the behavior of the product and has the following characteristics:

  - **Responsiveness:** Although the importance of performance varies between services, some are more critical than others, this project considers performance vital for the success and long life of this work. The same is said for providing answers and solutions to the stakeholders, always providing the best support possible, by listening to all their requests.

  - **Reliability:** The same as performance, all services need to be reliable to maintain and/or improve the clients confidence, this includes all adjacent services like billing.

    - **Technical competence:** Technical competence is an important factor for this kind of work, since the quality perception of services provided will be, mainly, determined on service availability, information accuracy, and response times, so not only the services need to be well designed, but the data should also be managed correctly and responsible.

    - **Flexibility:** As a consequence of the previous value based driver, well designed services also allow for better adaptability to change and problem solving, resulting in swifter answers to requests. However, like product customization it stays second to other, considered, more important drivers.

- **Relationship**

  The relationship relates to the services provided perception of the stakeholders.

    - **Image:** Reputation and credibility is directly connected to the image of his project. It's important to always give the feeling of confidence, control and dominance over the subject matter, even when the solution is saying no to something.

    - **Trust:** Like the Image value based driver, there needs to exist confidence between the client and the supplier. The supplier should keep with his promises and all information provided should be accurate, this being true with this project suppliers and clients.

    - **Solidarity:** Being able to receive help from suppliers and providing help to customer when there is a problem. This is important has it also contributes to the overall Image of the project.

    - **Time / Effort / Energy:** Quality usually requires effort, so it's implied that, at least in the beginning, there needs to be more time and effort dedicated to the project to guaranty its success.

    - **Conflict:** It's understandable that conflicting matters occur more in the beginning of the project's life cycle, but these must be resolved has soon has possible, the more these are allowed to continue the more they can hurt the project goals.

2.7.4. Canvas

The value networks shown in chapter 2.7.2 show a more detailed relationship between the stakeholders and the value to each of one, the table below intends to show a more general perspective overview into the MHP value proposition, infrastructure, customers and finances.

Table 8: Canvas

Key Partners	**Key Activities**	**Value Propositions**	**Customer Relationships**	**Customer Segments**
Dictionary providers Pharmacy suppliers Pharmacies	Software development Software maintenance Client support Data monitoring	Unification of technical protocols Unification of data Document archiving and management Provides access to historic data and statistics Protection of data Data agnostic	Support platform Technical support Proactive reporting	Pharmacies Pharmacy suppliers Laboratories Associations
	Key Resources		**Channels**	
	Human resources Equipment Software developed Virtual servers		Internet Associations Suppliers Regulators	

Cost Structure	**Revenue Streams**
Human Resources: 1 developers, 1 Commercial, 2 support staff IT Infrastructure: DigitalOcean / Amazon AWS	Revenue comes from communication usage and sponsorship; Sponsorship is provided by laboratories and suppliers and is intended audience is the pharmacies; Cost is not the same for all services, some are free and others vary accordingly to data consumption.

2.7.5. Competition analysis

In order to compare different solutions, the Analytic Hierarchy Process (AHP) was applied, created by professor Thomas L. Saaty in early 1970, to validate both quantitative and qualitative characteristics. The mains focus, in this analysis, is to prove that, to this project requirements, the best solutions is to create a platform specifically oriented to its needs instead of using existing platforms solutions (Jakupovic et al., 2010).

All of these solutions offer the ability to integrate systems and or APIs without writing code, but by transforming messages, for example with XSLT[33] or XQuery[34]. Of course, in any complex system, writing some code will still be necessary.

The following diagram shows a list of important criteria this system should have and a list of alternatives, for simplicity sake, all criteria are evenly important so that it's easy to compare them.

Note that some, maybe obvious criteria, were not included in this analysis because, it's not the purpose of this work to thoroughly analyze all solutions, these include, performance, stability, tools, technology aspects, analytics, quality and vendor time response. Also, platform aspects, like if the service runs in the cloud or what operating systems are supported is also ignored because this is not usually a concern when opting into a multi-service platform solution, i.e., this is a systems concern.

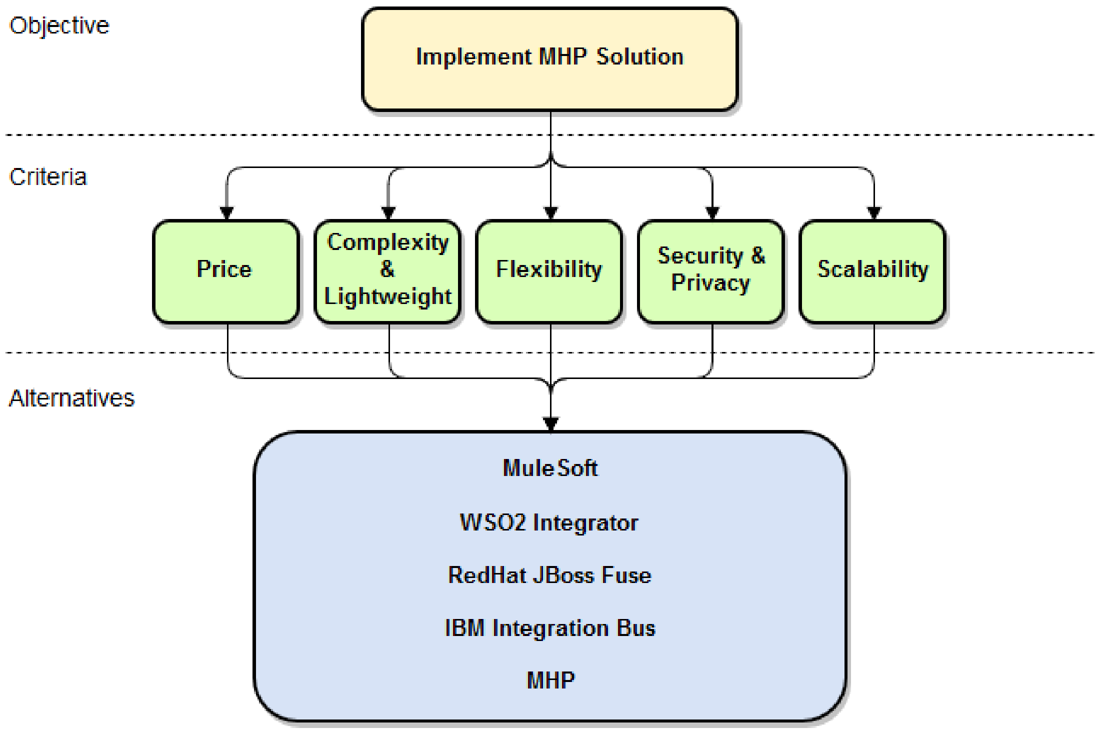


Figure 13: Analytic Hierarchy Process

- MuleSoft: Exists since 2006 and offers several solutions, manly integration platforms as a service (iPaaS), it's also known for offering "Mule ESB" which is an integration platform for companies to eliminate point-to-point integration development and Anypoint Platform which is a more general purpose platform for integrating service

33 https://www.w3schools.com/xml/xsl_intro.asp
34 https://www.w3schools.com/xml/xml_xquery.asp

and can be used as a cloud or on-premises based solution, it includes graphical interfaces to help manage the platform. Its solutions are not completely open-source. MuleSoft supports a community edition, but this edition is not considered here because it lacks important features, like for example, high availability (MuleSoft, 2017).

- WSO2 Integrator: This solution incorporates other WSO2 products like, WSO2 Enterprise Service Bus (WSO2 ESB), WSO2 Message Broker (WSO2 MB), WSO2 Data Services Server and WSO2 Business Process Server. So, it's a complete enterprise integration services solution. It's 100% open source (WSO2, 2017). This solution can be deployed on-premises, free solution, and on the cloud which is not free.

- Red Hat Jboss Fuse: For enterprise integration services Red Hat recommends JBoss Fuse. Like MuleSoft, Jboss Fuse started in 2006, all functionalities are open-source but the integration stack is very large to comprehend. It must be deployed on the JBoss Java EE web server (RedHat, 2014b).

- IBM Integration Bus: IBM includes in its integration package, B2B cloud services, gateway and transformation engines. It's a proprietary and partly open-source solution and it provides on-premises and cloud solutions (IBM, 2017).

- MHP: Micro Healthcare Platform is this work proposed solution. Its main focus is not a general purpose solution, but more oriented to this work requirements.

Next, using pairwise comparisons, the weight of each criteria is matched, so that it's clear which ones are more important than others. The scale used goes from 1 to 9, 1 meaning equal, 3 moderate, 5 strong, 7 very strong and 9 extreme, numbers in between are used for balance and compromise (Jakupovic et al., 2010).

Table 9: AHP – Criteria pairwise comparison

	Price	Compl. & Light	Flexibility	Security & Privacy	Scalability
Price	1	½	½	1/6	¼
Compl. & Light	2	1	1	1/6	1
Flexibility	2	1	1	1/6	1
Security & Privacy	6	6	6	1	6
Scalability	4	1	1	1/6	1

To be able to get a ranking of priorities from the pairwise comparison or matrix, it's first necessary to normalize the matrix, squaring the same and then calculate the eigenvector (to 4 decimal places), summing all rows and normalize the rows by dividing the row sum by the row totals. This process is repeated until the eigenvector is the same result has the previous iteration (through the process of experimentation this usually takes around 4 iterations).

The result is the following:

Table 10: AHP – Criteria solution

Criteria	**Score**	**Preference**	**Consistency** (product of Score vs Criteria)	**Consistency divided by Preference**
Price	0.058	4	0.2992	5.1586
Compl. & Light	0.1097	3	0.566	5.1595
Flexibility	0.1097	3	0.566	5.1595
Security & Privacy	0.5906	1	3.0476	5.1602
Scalability	0.1321	2	0.682	5.1628
λmax (average)				5.1601

The previous (final) eigenvector gives us the score for each criteria, the bigger score the better. For this case Security & Privacy is the most important criteria.

To prove that the values are consistent, the Consistency Index (CI) must be calculated:

$$CI = (\lambda max - n) / (n - 1) \tag{2}$$

CI = (5.1601 - 5) / (5-1) = 0.04

According to Thomas Saaty's table, the Random Consistency Index is: RI(4) = 0.90

Table 11: AHP – Thomas Saaty's table

m	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
RI	0	0.58	0.90	1.12	1.24	1.32	1.41	1.45	1.51

Finally, the Consistency Ratio (CR) can be calculated:

$$CI = (\lambda max - n) / (n - 1) \tag{3}$$

CR = 0.04 / 0.90 = 0.0444

Since the CR, 0.0444, is lower than 0.1, it can be assumed that the values of the weight of each criteria and the eigenvector are consistent.

The previous process is repeated for all the alternatives under each criteria, resulting in an eigenvector for each alternative vs criteria.

In each of the following criteria is explained how the evaluation was performed, and also, when possible, it was also used third party evaluation on the same solutions, like for example Gartner's software and service reviews (Gartner, 2017).

2.7.5.1. Price

In order to be easier to compare the alternatives the estimated price per year was considered for each solution. These prices must be viewed as an approximation to the actual cost, since they usually change often and can also vary depending on the commercial agreement between the suppler and the customer.

WSO2 Integrator is the only free of the four platforms solutions, but charges for support which is included in the other solutions commercial licenses. The support options are many and not simple for someone who is approaching this solution a first time. The cloud version however, is a pay solution and the costs of this solution is easier to control and scale, so instead of considering the support costs, which would actually be more expensive, the cost bellow corresponds to the cloud version (WSO2, 2017).

MuleSoft has a community edition which is free, but because it's limitations it's not considered here. For MuleSoft the only way to get a price evaluation was to contact a sales representative.

IBM Integration Bus and RedHat Fuse only have commercial licenses, but provide trial-periods for development and testing purposes (IBM, 2017) (RedHat, 2017).

MHP does not have a direct cost associated since is developed in-house, but it has a cost of development and maintenance, of course all the other solutions also have a cost of integration, learning curve and maintenance, so as an estimate, the cost of the MHP has been determined to be a full-time developer, the deployment costs are not considered because, in comparison with the other solutions a cost p/core would be only 5€ like it's possible to see in chapter 2.4.1.4.

Table 12: AHP – Price per year (WSO2, 2017) (RedHat, 2017) (IBM, 2017a/b)

Alternatives	Score	Preference
MuleSoft	6 000 € p/core	4
WSO2 Integrator (WSO2, 2017)	3 576 € p/core	3
RedHat Fuse (redhat, 2017)	2 718 € p/core	2
IBM Integration Bus (IBM, 2017a/b)	14 208 € (starter value)	5
MHP	1 500 €	1

Table 13: AHP – Price pairwise comparison

	MuleSoft	WSO2	Fuse	IBM	MHP
MuleSoft	1	0.5	0.3334	2	0.25
WSO2 Integrator	2	1	0.6668	4	0.5
JBoss Fuse	3	1.4997	1	5.9988	0.7499
IBM Integration Bus	0.5	0.25	0.1667	1	0.125
MHP	4	2	1.336	8	1

Table 14: AHP – Price solution

Alternatives	Score	Preference
MuleSoft	0.0952	4
WSO2 Integrator	0.1905	3
JBoss Fuse	0.2857	2
IBM Integration Bus	0.0476	5
MHP	0.381	1

2.7.5.2. Complexity & Lightweight

Complexity is measure in terms of supported technology, configuration and maintainability. For each of the solutions presented here, the requirements for setting up a minimum viable product were analyzed following the official documentation of each solution.

JBoss Fuse and MuleSoft both use Drools www.drools.org) for its Business Rules Management System (BRMS) and jBPM (www.jbpm.org) for its Business Process Management, although

JBoss Fuse supports newer versions of the protocols, 6.0 instead of 5.0 for BRMS and 6.0 instead of 4.4 for the jBPM, also in the case of jBPM it means that only Jboss Fuse supports BPMN 2.0.

IBM has its proprietary solutions, it uses Operational Decision Manager (ODM) for its BRMS and BPM (Jackson, 2016).

WSO2 Integrator is powered by the Activity BPMN Engine and Apache Orchestration Director Engine (ODE) BPEL engine, meaning business process can be written in BPMN 2.0 standard or WS-BPEL 2.0 standard. (WSO2, 2017).

To organize service integration JBoss Fuse relies on the open standard Service Component Architecture (SCA) (IBM, 2017) which supports components like Camel, business process execution language (BPEL), BPMN and rules or events. MuleSoft requires external proprietary service assembly process.

All solutions, except for the MHP Platform, support GUIs to facilitate basic configuration, but to support more complex requirements, it's always necessary to dive into code and configuration files, for this reason the GUIs are not considered to represent a major difference, since almost all requirements of this work imply complex configurations. Of course, once all is set up, GUIs allow for a graphical visualization of the system and this is not without its value.

According to Gartner[35], in the category of Ease of Development, the score in a scale of 5 is as follows (Gartner, 2017):

- MuleSoft: 4.3
- WSO2: 3
- RedHat Fuse: 3.8
- IBM: 3.6

The above analysis suggests that MuleSoft Fuse in a better choice when thinking in terms of complexity and lightweight.

MHP solution could be easily considered the easiest to work with in this scenario since this work is its development, but it's important to also considered when new developers need to be integrated into the project, for this reason, the MHP solution is considered on equal grounds with MuleSoft in this category.

35 https://www.gartner.com/reviews/market/Full-Life-Cycle-API-Management

Table 15: AHP – Complexity & Lightweight pairwise comparison

	MuleSoft	**WSO2**	**Fuse**	**IBM**	**MHP**
MuleSoft	1	4	2	3	1
WSO2 Integrator	0.25	1	0.5	0.75	0.25
JBoss Fuse	0.5	2	1	1.5	0.5
IBM Integration Bus	0.3333	1.3333	0.6667	1	0.3333
MHP	1	4	2	3	1

Table 16: AHP – Complexity & Lightweight solution

Alternatives	**Score**	**Preference**
MuleSoft	0.3243	1
WSO2 Integrator	0.0711	4
Fuse ESB	0.1622	2
IBM Integration Bus	0.1081	3
Build Your Own	0.3243	1

2.7.5.3. Flexibility

Flexibility is measure in terms of adaptability, meaning that features like compatibility, license flexibility, time to develop and ease of integration with external APIs are considered here.

JBoss license is in the form of a middleware subscription meaning that it's not limited to one software product (redhat, 2017), WSO2 cloud subscription is also unified (WSO2, 2017), MuleSoft and IBM enterprise license is different for each product, adding less flexibility in the license agreement (MuleSoft, 2017).

MuleSoft, WSO2, redhat and IBM solutions all support HL7 which is important since this work is directed to the healthcare market.

MHP has the biggest advantage here since it can be whatever this project wants it to be.

According to Gartner, in the category of Integration & Deployment, the score in a scale of 5 is as follows (Gartner, 2017):

- MuleSoft: 4.3
- WSO2: 3

- redhat Fuse: 3.8
- IBM: 3.5

Table 17: AHP – Flexibility pairwise comparison

	MuleSoft	WSO2	Fuse	IBM	MHP
MuleSoft	1	4	2	3	0.5
WSO2 Integrator	0.25	1	0.5	0.75	0.125
JBoss Fuse	0.5	2	1	1.5	0.25
IBM Integration Bus	0.3333	1.3333	0.6667	1	0.1667
MHP	2	8	4	6	1

Table 18: AHP – Flexibility solution

Alternatives	Score	Preference
MuleSoft	0.2449	2
WSO2 Integrator	0.0612	5
Fuse ESB	0.1225	3
IBM Integration Bus	0.0816	4
MHP	0.4898	1

2.7.5.4. Security & Privacy

In terms of security, all software products support basically the same solutions, all support HTTPS, authorization and authentication.

Privacy is another matter, all solutions support on-premises deployments, meaning, privacy is more of a concern of the client implementing the solution, if the cloud option is preferred then privacy is subject to how much trust is giving to the cloud provider since the data would not be protected from the cloud provider.

Since the MHP solution is the only with guaranties that data will not be viewed by any third party it gains more points in this category.

Table 19: AHP – Security & Privacy pairwise comparison

	MuleSoft	**WSO2**	**Fuse**	**IBM**	**MHP**
MuleSoft	1	1	1	1	½
WSO2 Integrator	1	1	1	1	½
JBoss Fuse	1	1	1	1	½
IBM Integration Bus	1	1	1	1	½
MHP	2	2	2	2	1

Table 20: AHP – Security & Privacy Solution

Alternatives	**Score**	**Preference**
MuleSoft	0.1667	2
WSO2 Integrator	0.1667	2
Fuse ESB	0.1667	2
IBM Integration Bus	0.1667	2
MHP	0.3333	1

2.7.5.5. Scalability

Scalability is measure mainly in terms of horizontal growth, only with horizontal growth can a solution be quickly, easily and cost-effective scaled.

The MHP solution is only solutions with 100% horizontal scaling, all other solutions support some sort of enterprise high-availability but all with adjacent complexity and costs.

Table 21: AHP – Scalability pairwise comparison

	MuleSoft	WSO2	Fuse	IBM	MHP
MuleSoft	1	1	1	1	½
WSO2 Integrator	1	1	1	1	½
JBoss Fuse	1	1	1	1	½
IBM Integration Bus	1	1	1	1	½
MHP	2	2	2	2	1

Table 22: AHP – Scalability solution

Alternatives	Score	Preference
MuleSoft	0.1667	2
WSO2 Integrator	0.1667	2
Fuse ESB	0.1667	2
IBM ESB	0.1667	2
MHP	0.3333	1

2.7.5.6. Final result (AHP)

For the final result, a matrix with the eigenvectors results from the previous step is squared with the eigenvector from the criteria, the bigger value from the result is the best choice.

Table 23: AHP – Criteria with alternatives pairwise comparison

	Price	Compl. & Light	Flexibility	Security & Privacy	Scalability
MuleSoft	0.0952	0.3243	0.2449	0.1667	0.1667
WSO2 Integrator	0.1905	0.0811	0.0612	0.1667	0.1167
JBoss Fuse	0.2857	0.1622	0.1225	0.1667	0.1667
IBM Integration Bus	0.0476	0.1081	0.0816	0.1667	0.1667
MHP	0.381	0.3243	0.4898	0.3333	0.3333
Criteria Eigenvector	0.058	0.1097	0.1097	0.5906	0.1321

Table 20: AHP – Final Solution

Alternatives	Product with Criteria eigenvector	Preference
MuleSoft	0.1884	2
WSO2 Integrator	0.1471	4
Fuse ESB	0.1683	3
IBM ESB	0.144	5
MHP	0.3523	1

The final result proves that the MHP solution if the preferred choice under these criteria and alternatives.

2.7.5.7. Final thoughts

A platform without documentation would be pretty difficult to maintain, so this was also verified in all solutions. Documentation capabilities was verified in two parts, documentation for the base platform and documentation for the develop integration APIs. All solutions seem to value documentation and include documentation for its platform base functionalities, also, they provide functionality to document all integration APIs. The MHP solution developed in this work also addresses this (see chapter 3.11.2).

It's also important to note that nothing substitutes real experience with each of these platforms, meaning that it someone with, more or less three years experience, with any of these platforms could probably had a different perspective on this evaluation.

2.8. SWOT Analysis

To identify the main aspects that characterize the market strategic position, being externally or internally, a SWOT analysis is used. It's important to note, however, that this analysis is valid for a specific moment in time and that strategic management should be a continuous process.

Table 24: SWOT Internal Analysis

Strengths	**Weaknesses**
Technical Know-How; Technology used has already proven to be efficient and stable; No complexity in logistics; Technical team motivation, since the development of this project is done in emerging technologies; Requirements are specified and verified by our partners before reaching all the clients.	Large initial investment; Commercially not very oriented; In its initial stage.

Table 25: SWOT External Analysis

Opportunities	**Threats**
Lack of immediate competition; High complexity of today's services implementation generates need to simplify technology requirements;	Project openness can lead to competition; Regulatory mandates can mute or limit project evolution.

3. Development and Implementation

This chapter describes this work development approach and its implementation, from a high-level perspective as well has a more granular description when needed.

3.1. Developing model

This chapter gives an overview of the more relevant concepts, techniques and dependencies that are used during the planning and the development stage of a micro-service architecture.

3.1.1. Essential Dependencies

All services in this platform where developer as a Play Framework instance, the framework, by default, already incorporates the most basic dependencies, but because this is designed to be a lightweight framework, it only comes with the essential, so, some dependencies that are necessary for this platform need to be added. The following table shows a list of the dependencies that are essential to all services as well as its version. Since this project uses SBT[36] (Simple Build Tool) has the build tool, the dependencies shown here use the SBT syntax.

Specific service dependencies are described in it’s appropriate section in this document, not including these dependencies.

36 http://www.scala-sbt.org/

Table 26: Platform essential dependencies

Dependency	Description
"org.scalatestplus.play" %% "scalatestplus-play" % "3.1.1" % Test	Testing framework for automated testing
"com.typesafe.play" %% "play-json" % "2.6.5"	JSON library, for working with JSON messages
"com.typesafe.play" % "play-ahc-ws-standalone_2.12" % "1.1.0", "com.typesafe.play" % "play-ws-standalone-json_2.12" % "1.1.0"	Play Framework essentials for web services client operations and JSON support
"com.typesafe.play" %% "play-mailer" % "6.0.1",	Email operations, All services need to report errors.
"com.typesafe.akka" %% "akka-cluster" % “2.5.4”, "com.typesafe.akka" %% "akka-cluster-metrics" % “2.5.4”,	Akka additional dependencies. Play already comes with basic Akka support, but Cluster and Metrics support must be manually added.
"org.abstractj.kalium" % "kalium" % "0.7.0"	Encryption library. All services need this because the standard status messages that this platform uses needs to encrypt some info.
mhp-sys	This is a custom developed library, developed in Scala for this project, and its purpose is to fulfil cross cutting concerns across the platform different apps. At the moment its main function include encryption functions, and more general system data functions. This library is not added through SBT but rather as a local library. Play framework will include any library that is simply added to the *“lib”* folder inside each Play app.

3.1.2. Events-First Domain-Driven Design

This concept is from Russ Miles, it helps developing distributed systems by creating an abstraction on the nouns (objects names) and verbs (the events) and the idea is to focus on the things that happen in the system and only then worry about structure, this gives a different approach from more traditional object-oriented programming (OOP) and domain-driven design (DDD). It helps understand how things flow in the system and by consequence have a temporal perceptive on the system (Bonér, 2017).

Also by focusing on the events it helps in creating Bounded Contexts[37] which by themselves help in separating concerns and complexities of a section of the system (Bonér, 2017).

3.1.3. Stateless Web Tier

A stateless architecture allows the scalability of components, if a component is scalable then it's possible to replicate the same between different nodes, and access by the consumers can be seamless distributed (Hayes, 2011).

Since Scalability is a major concern all endpoints should be made stateless unless that a stateful approach is essential, like for example in WebSockets connection, but even in these endpoints the load-balancing capabilities must be maintained.

3.1.4. Non-Blocking

Services and threads spend most of the time idle, waiting for data from other services and data stores, this is because I/O is very expensive. The Play's web service library[38] can be used for making non-blocking calls, for example:

```
def someMethod() = Action.async {
    val future = ws.url("http://www.isep.ipp.pt.com").get

    future map {
        response => Ok(response.body))
    }
}
```

Code 2 – Example code of non-blocking operation

The ws.get() method returns a Future of a Response (Future[Response]) which is a Scala Future that will eventually contain the Result.

3.1.4.1. Non-Blocking SOAP calls

SOAP requests are typically blocking I/O, this is because, internally, Java default implementation of JAX-WS uses blocking HTTP calls[39].

The solution is this case is to use another JAX-WS implementation, in this case the Apache Cxf[40] library is used which can use non-blocking HTTP calls. To do this there are two dependencies that need to be added, cxf-rt-frontend-jaxws and cxf-rt-transports-http-hc, the

37 https://martinfowler.com/bliki/BoundedContext.html
38 https://www.playframework.com/documentation/2.6.x/ScalaWS
39 http://cxf.apache.org/docs/asynchronous-client-http-transport.html
40 http://cxf.apache.org/

first one makes CXF the JAX-WS default implementation, and the second allows CXF to make non-blocking HTTP client calls.

To test that this is working Apache Benchmark[41] (ab) tool can be used, for example:

```
ab -c 50 -n 1000 localhost:9000/async
```

Code 3 – Example of Apache Benchmark tool

The previous command makes 1000 requests 50 at a time, meaning 50 concurrent requests, to an endpoint (localhost:9000/async). During the tests another tool like *visualVM* or *Netbeans Profiler* can be used to see the working threads.

3.1.4.2. Future composition

Future composition can be achieved very easily in Scala and in functional programming in general, it's easy to accumulate the *map* function concatenating future's on after the other, but to make things easier to read and maintain, Scala has the for comprehension with yield pattern[42], and it's also a way to avoid the *callback hell*[43]. With for-yield syntax futures can follow other futures and use their outputs like in the following example:

```
try {
  for {
    // 1. Validate authentication
    authenticated <- function01()

    // 2. Relay message to backend service
    backend <- function02(authenticated)

  // 3. process the results
  } yield processResponse(authenticated, backend)

// 4. deal with exceptions
} catch {
  case e: Exception =>
    log.error("System error", e)
}
```

Code 4 – Example code of future composition with for-yield syntax

1. First a function is called to authenticate the request, this function returns a future with the result, *Future[Result]*, when the future completes it then call the second function;

41 http://httpd.apache.org/docs/current/programs/ab.html
42 https://docs.scala-lang.org/tour/sequence-comprehensions.html
43 https://engineering.linkedin.com/play/play-framework-async-io-without-thread-pool-and-callback-hell

2. The second future calls a function which also returns a future and passes the previous result;

3. After all futures are completed their return values can be processed in the yield portion of the code;

4. Any exception thrown can be catch in the final block of code.

Using this method, dealing with futures is easier and very readable and it's possible to see that even if the code grows it will still maintain a readable structure.

It's also important to note the following:

- This is all non-blocking code;

- The exceptions ca be caught for each future using the *recover method* after each future;

- The futures are called sequentially, but can also be called in parallel, for this all that is needed is to define the futures outside the for comprehension, like in the following example.

```
val authenticated = function01()
val backend  = function02()

for {
  result1 <-  authenticated
  result2 <-  backend
} yield processResponse(result1, result2)
```

Code 5 – Example code of future composition with for-yield syntax in parallel

3.1.5. Data-Access

Data access is typically a blocking operation (see Figure 4), for example, methods in the JDBC to access a database are all blocking, so to have a reactive application this needs to be addressed, fortunately the Play framework has mechanisms for this, for example:

```
def save(hash: String, fullUrl: String)(implicit ec: ExecutionContext):
Future[Unit] = Future {
  DB.withConnection { implicit c =>

    val sql = SQL("insert into shorturls (short, fullurl) value ({short},
{fullurl})").on("short" -> hash, "fullurl" -> fullUrl)

  sql.executeInsert()
}
```

Code 6 – Example code of non-blocking data access

Although the example above is a working approach, it's always better, whenever possible, to use a database model that is asynchronous from the start, this is the case of MongoDB for which Play's own team help develop a non-blocking driver called ReactiveMongo[44].

The library used for communicating with the database also has an impact on the developing model. Some libraries, like Slick[45], are by design asynchronous, while other like Anorm[46] are not. If a library is not asynchronous by design it's up to the developer to guaranty that calls to the database are made asynchronous whenever needed, like in the previous example.

3.1.6. 2-Way Reactive Model

Play's 2-way reactive model means that it doesn't acknowledges (ACK) the received transport data until it's ready to process it, for example, if a client sends some data and the same is being relayed to another service, the message received from the client will only be acknowledged when the server is ready to process more data, this way, Play facilitates the way to achieve reactive in all intervened entities.

This is a very important feature and one the typically is only achieved by applying backpressure techniques, this is not to say that backpressure is implement by default, it still exists the need to control backpressure, for example, in streaming scenarios, but it helps substantially the developer in already providing this kind of functionality.

3.2. Cross Cutting Concerns

Cross cutting concerns in this platform can be resumed to logging, monitoring and metrics gathering, there's more than one approach to deal with these concerns as is explained in the following sub-chapters.

3.2.1. Filters Vs Action Composition Vs External Action

An Action in the Play Framework is basically a function that catches a request and produces a response. Action composition in this context means that it's possible to create a class that extends the basic action functionally and add the necessary functionality, this new class is then used to catch the request (Lightbend, 2017).

Filters are another way to catch, and possible change, all requests, typically filters are used for logging, collecting metrics, compression and security (Lightbend, 2017).

Basically, filters should be used when an action is intended to be applied to all requests, and action composition, when just some specific requests should be affected (Lightbend, 2017).

44 http://reactivemongo.org/
45 https://www.playframework.com/documentation/2.6.x/PlaySlick
46 https://www.playframework.com/documentation/2.6.x/ScalaAnorm

3.2.2. Logging

Logging in the Play Framework, by default uses the Logback[47] framework. It natively implements the SLF4J API[48] to facilitates the migration, if needed, from Logback to another logging framework. This platform uses this framework to log all intended messages, from debug to error messages, in the file system. Each file has the duration of a day, after which is compressed, archived and saved for 30 days, note that these settings can be changed by editing the *logback.xml* configuration file, but these are the settings that this platform uses.

Typically, every logging message must be declared explicitly, but there are some cases where it's useful to automate the logging without needing to declare every message, such is the case, for example, for all request that the platform receives, in which is important and intended to record all incoming and outgoing messages. To accomplish this a filter is used:

```
class LoggingFilter @Inject()(implicit val mat: Materializer,
                              ec: ExecutionContext) extends Filter {

  val log = LoggerFactory.getLogger("LoggingFilter")

  def apply(nextFilter: RequestHeader => Future[Result])
           (requestHeader: RequestHeader): Future[Result] = {

    val startTime = System.currentTimeMillis

    nextFilter(requestHeader).map { result =>

      val endTime = System.currentTimeMillis

      val requestTime = endTime - startTime

      log.info(s"${requestHeader.method} ${requestHeader.uri} took $
{requestTime}ms and returned ${result.header.status}")

      result
    }
  }
}
```

Code 7 – Filter to log all incoming and outgoing messages

The apply method in the previous code, which is a curried function, as two parameters, the first one, *nextFilter*, is a function that takes a request header and produces a result, the second, *requestHeader*, is the actual request header. Inside this method the start time is saved, then the nextFilter function is called, which calls the actual intended action function for this request, when the request processing completes, the end time is saved and compared to the start time to log how much time each request took.

47 https://logback.qos.ch/
48 https://www.slf4j.org/

An alternative to this is to use a logging decorator, which, in this context, basically means an Action class that extends the basic Play Action functionalities, to log just specific requests, but in this case the intended purpose is to log all requests without exceptions.

3.2.3. Monitoring

Monitoring can be achieved in several ways, ones better than others, but usually the better ones have a commercial license cost. There are open source monitoring tools, like Kamon (http://kamon.io) that can be used to capture Akka metrics and can be integrated into the play framework but these don't include visual presentation layers.

Since integrating a commercial license would increase the cost of the platform and add a commercial dependency it is not considered for this version of the platform, other tools, like Kamon, would add complexity to the platform without an easy way to consume the results, for these reasons, for this version of the platform, monitoring is restricted to external systems that monitor the main resources and to in-code monitoring by supervising the actors.

As for supervisor strategy, the platform does not use "all-for-one strategy", it only uses "one-for-one strategy", for simplicity reasons, its recommended by the Akka team and it's easier to maintain an architecture where the supervisor only reacts to the children that failed.

Monitoring main resources is obtained at the systems level and not at the software level, meaning that there are services specifically for this purpose and are setup as follows:

Table 27: Monitoring metrics

Role	Description
Bandwidth - Inbound	Above 75 Mbps for 5 min
Bandwidth - Outbound	Above 75 Mbps for 5 min
CPU	Above 75 % for 5 min
Disk — Read	Above 75 Mb/s for 5 min
Disk — Write	Above 75 Mb/s for 5 min
Disk Utilization	Above 75 % for 5 min
Memory Utilization	Above 75 % for 5 min

3.2.4. Metrics

Metrics can be obtained by monitoring the services externally and/or internally:

- External: metrics are created based on the response of the API calls, for example, how many requests per second can be achieved a certain request and circumstance;

- Internal: metrics code is added together with business code so that it can record a higher level of detail.

Although the internal method can obtain more detail, it also adds more complexity to the code, for this reason this work uses the external approach has a means to obtain the more essential metrics. The internal approach can still be used in the future, for example, for some more specific metrics requirements, in this case, however, it's advisable to use AOP (Aspect Oriented Programming) instead of just mixing the business code with the metrics code, this way the complexity of the metrics code can be abstracted from the business code.

3.3. Security

Like previously stated, security is a major concern, this chapter addresses the measures implemented to make the platform a secure environment. Security is also a continuous concern, the next chapters describe the essential but should not be taken as complete solution.

3.3.1. HTTPS

All services use HTTPS, HTTP is disable by default, this is the only way to keep communication private, for these certificates are needed to ensure that not only a TLS connection is established but also that this platform consumers can verify the identity of it's provider as trustworthy.

For certificate issuing and renewing the initiative Let's Encrypt[49] is used, this gives this platform a free, public recognized, certification authority solution and it also means that it's possible to automate the renewing of certificates. It's very important to automate this process since all the certificates issued have a 90 day expiration period. There are already several tools online that already integrate Let's Encrypt[50] with popular web servers, but they typically only work in Linux environments and, at the moment, none cross-platform tool exists to integrate with Plays Framework.

So, to accomplish the automatic renew of certificates a small Java tool was developed, this tool has two function modes, one to produce new certificates and another to renew them.

49 https://letsencrypt.org/
50 https://letsencrypt.org/docs/client-options/

For example, to emit and renew new certificates the commands are:

Table 28: Example commands to create and renew certificates

Create / Renew	Command / Arguments
Create new Certificates	Java -jar mhpCertTool.jar –**domains**=domain.com,app.domain.com --**operation**=create --**staging**=0 --**country**=Portugal --**local**=Porto --**organization**=LTS --**organizationalUnit**=Project --**user**=user1 --**pass**=p@ass
Renew Certificates	java -jar mhpCertTool.jar --**domains**= domain.com,app.domain.com --**operation**=renew --**staging**=0 --**user**= user1 --**pass**=p@ass

The following table describes the commands of the MHP Certification Tool:

Table 29: MHP Certification Tool arguments

Argument	Description
--domains	The domains to create or renew, comma-separated.
--operation	create / renew. To create or to renew certificates.
--staging	0 / 1. 0 = production; 1 = testing.
--country	Certificate Country attribute tag.
--local	Certificate Local attribute tag.
--organization	Certificate Organization attribute tag.
--organizationalUnit	Certificate Organizational Unit attribute tag.
--user	The username (If the username does not exist it will be automatically created as a new user).
--pass	The users password.

This tool has the following dependencies:

Table 30: MHP Certification Tool dependencies

Dependency	Description
ACME4J[51]	For an easy integration with the ACME protocol of Let's Encrypt.
OpenSSL[52] tool	To create a PKCS12[53] file from the certificates.
Java Keytool[54]	To create the Java Keystore from the PKCS12 file.

When creating certificates, the tool will ask for a DNS entry confirmation, which is the default method of domain validation, it's hard-coded but can be changed if necessary, this is only done one time, after that renewing is automatic, this tool also packages the certificates in a Java KeyStore which is the default Java way to access certificates, and is also the way of Play Framework. The only thing left to do is to copy the file to the Routers (see chapter 3.10) components folders to keep the certificates updated.

3.3.2. CORS, CSRF, XSS and SQL Injection

- CSRF

Cross Site Request Forgery (CSRF) (RFC 6749 chapter 10.12) is a security exploit where an attacker tricks a victim's web application into making a request for him using the victims session containing his session token which can be used for authentication, in other words the victims web application makes unwanted requests in its name (Lightbend, 2017) (OWASP 2017). Preventing CSRF attacks is not easy and there is not an easy, one-time solution, to completely prevent all CSRF attacks, but this can be mitigated and the attack surface reduced substantially (Shahriar et al., 2010).

Since it is not a priority of the platform to host web-sites, CSRF attack on web-sites is not a major concern, but, CSRF attacks can also target authorization mechanisms and so it's important to include protection against these attacks. In a recent, 2017, report by, Sudhodanan et al., 132 top web sites were testes and the conclusion was that 72% of them had vulnerabilities, including web sites from Microsoft, Google and eBay, proving the previous statement that securing against CSRF is not an easy task.

Play framework has integrated mechanism to help with CSRF attacks, it can be configured by setting cross-origin policies and, by default, it requires a CSRF check when a request is not GET, HEAD or OPTIONS, the request has one or more Cookie or Authorization headers (otherwise no session based authorization data is being sent) or the CORS filter is not configured to trust the request's origin. CSRF is checked by Play by placing a CSRF token in the

51 https://github.com/shred/acme4j
52 https://www.openssl.org/
53 https://www.ssl.com/how-to/create-a-pfx-p12-certificate-file-using-openssl/
54 https://docs.oracle.com/javase/6/docs/technotes/tools/solaris/keytool.html

query string or body of the requests and in the user’s session, then it compares both to see if they match, if they do not match, an access forbidden (403), error is thrown (Lightbend, 2017).

- CORS

Cross Origin Request Sharing (CORS) (RFC 6454), is the definition to when a request is made to a resource in a different domain, protocol or port from its own. This is a standard practice, for example, when loading resources like images or other static resources from external domains. This standard, basically, works by letting the servers add new HTTP headers that specify the origins that can access local resources.

In a micro-services environment, where, a request can involve several services to communicate with each other, CORS needs to be addressed and set up in such a way that each service can communicate with each other and still maintain a high level of security.

In the case of this platform, the router needs to be open to CORS requests since its primary objective is to filter and proxy requests from various clients, has for the other services they CORS is configured so that is restricted to requests from only the services that they need to communicate with, for example, all services need to allow requests from all Router instances and MTS instances, but they don’t, necessarily, need to communicate with other services.

- Cross Site Scripting

Cross Site Scripting[55] (XSS) is a client-side vulnerability where an attacker can inject code into a website. Plays template engine[56], already provides integrated protection against XSS attacks.

- SQL Injection

SQL Injection[57] is an attack that consists in injecting unintended code into an SQL command, this kind of attack can be prevented by curating the dynamic code in SQL calls, like stored procedure arguments. The Anorm[58] library that is used in this project already provides the mechanism that apply protection for this kind of attack.

- More information

It’s not the purpose of this work to dive much deeper into the above security concerns, but the OWASP Testing Guide[59], currently at version 4, has many more valuable information and it’s a helping guide on how to test and prevent these kind of attacks.

55 https://www.owasp.org/index.php/Top_10_2013-A3
56 https://www.playframework.com/documentation/2.6.x/JavaTemplates
57 https://www.owasp.org/index.php/SQL_Injection
58 https://playframework.com/documentation/2.6.x/ScalaAnorm
59 https://www.owasp.org/index.php/OWASP_Testing_Guide_v4_Table_of_Contents

3.3.3. Authentication and Authorization

Like previously stated, there isn't allowed any unauthenticated request in the platform, so inside the HTTPS connections every request must be validated. The method by which they are validated varies if it's a request to the business API of it it's a request to the Web UI.

All internals API calls use JWT (RFC 7519) authentication and the Web UI uses Cookie authentication, basically the only reason Cookie based authentication is used is because, at the moment, it's more compatible with browsers than JWT authentication, and since authentication is mandatory, this affects the time to market of the entire solution.

The next two tables present some of the pros and cons of each authentication method.

Table 31: Pros and cons of cookie based authentication[60]

Pros	Cons
Small network throughput on client side; Very compatible with traditional browsers; Client fingerprinting; Can be stateless or stateful; Can use "remember me" functionality.	If stateless, has bigger network throughput on client side; If stateful, has bigger network throughput on the server side; It needs a backing store to be stateful; Not has ideal as JWT for SPAs or mobile apps; If not protected, can be vulnerable to CSRF attacks; Not very compatible with CORS.

60 See https://www.silhouette.rocks/v5.0/docs/authenticator and https://auth0.com/blog/cookies-vs-tokens-definitive-guide/

Table 32: Pros and Cons of JWT based authentication

Pros	Cons
Very compatible with SPAs and mobile apps;	Larger network throughput on client side;
Can be stateless;	Larger network throughput on the server side (if backing store is used);
Not vulnerable against CSRF attacks (since the browser doesn't automatically add the header to your request);	Less than ideal for traditional browser based websites;
Plays well with CORS;	No client fingerprinting;
Can transport arbitrary claims;	If stateless, the token cannot be invalidated.
Can be used for "remember me"[61] functionality.	

From these two tables is possible to see that the advantages of JWT really outweigh the advantages of the cookie approach, and even the disadvantages of JWT are not a very big concern for this platform.

To unify the development of the authentication mechanisms and facilitate it's continuous maintenance, a stable and purpose focused framework is used, Silhouette[62] is a framework developed specifically for the Play Framework and Scala, although future versions will be able to be used has an independent component, it supports several authentication methods and allows for an easy switch between then without the need to change the business logic of the code. Its fully asynchronous and follows the OWASP Authentication Cheat Sheet[63]. At the moment this framework is used to implement JWT and Cookie based authentication.

Being a major concern and responsibility, authorization and authentication is implemented in its independent micro-service and persistence architecture (see chapter 3.11).

For simplicity reasons in the rest of this document, when referring to the authentication/authorization service it will simple be referred to has authorization service since the authorization implies authentication and the other way around is not always true.

61 The same token can be used in several requests for a pre-determined period of time.

62 https://www.silhouette.rocks

63 https://www.owasp.org/index.php/Authentication_Cheat_Sheet

3.3.4. Firewalls

It's not only service that should be single focused in its responsibilities, this also applies to Firewalls. By having each service protected by its own firewall protection is assured both from outside traffic and from inside traffic.

This also has the added bonus that configuring, managing and troubleshooting the firewalls because an easier task since each firewall is a smaller component compared to a perimeter firewall with all the rules in one place.

3.3.5. Data Protection

To better prepare for the changes ahead (see chapter 2.6), this platform implements the necessary implementation designs to better protect the data both from access and/or tampering.

Besides authentication and authorization there are some steps that are taken to achieve this:

- **Configuration Data**: files with sensitive information, like credentials, are not included in version control. Sensitive information is saved in a safe environment, like a password manager store (in this case KeePass[64]). During development, testing and staging, separate files uses just for development are used with test data, production files are keep protected and the services are configured so that they know when to use one file or the other. For example, during development a configuration file "application.conf" is used, when the service is started in production the same file is loaded but another file "application.prod.conf" is also loaded overwriting the previous file. The production file only overwrites the same keys, meaning the production file doesn't need to have all configurations, but just the ones that are meant to be different in production.

- **Logging and Monitoring:** All request to the platform are recorded for possible reviewing and all access validations are recorded separately to normal request. This means that all services are configured to record access related data in a separate files and format. This has the advantage that access data can be better protected than normal service usage data and that reporting any access violations also is separate from normal usage reporting. Even inside the organization different access can be granted to different personnel, to the access data, making it more secure (example configurations on annex 6.6).

- **User Data:** Sensitive user data, like credentials, are saved in a database always with a hashing algorithm (SHA-2 of 256 bits). To achieve this the library Kalium[65] is used, which is a Java library based on the popular open source library Libsodium[66].

64 https://keepass.info
65 https://github.com/abstractj/kalium
66 https://github.com/jedisct1/libsodium

Libsodium needs to be installed in the Operating System where the service is running in order for Kalium to find it, for this, there already exists pre-compiled binaries available[67].

3.3.6. End-to-End Encryption

End-to-end encryption is achieved by sharing a Key and a Protocol between the consumers and the providers, for example when a pharmacy requests product prices from a supplier, this information should stay between the pharmacy and the supplier. Basically, the data is encrypted on one side with the key and protocol, the platform only sees the data in the encrypted format, and then it's send to its destination where it is decrypted with the predetermined protocol and key.

Encryption is optional, and, for the moment, this platform only provides the instructions and procedures to implement the correct encryption protocol in order to achieve end-to-end encryption. The platform itself does not encrypt or decrypt the data.

Table 33: Encryption protocol parameters

Parameter	Value	Description
Password hashing algorithm	PBKDF2WithHmacSHA1	. PBKDF2 stands for Password-based-Key-Derivate-Function, and is responsible for hashing the password or key along with a salt value. This process is repeated N number of times and the resulting key can be later used as a cryptographic key. . HMAC stands for Keyed-Hash Message Authentication Code, and is responsible for calculating the message authentication code (MAC). It uses a cryptographic hash function combined with the cryptographic key (created by PBKDF2). The HMAC process mixes a secret key with the message data, hashes the result with the hash function, mixes that hash value with the secret key again, and then applies the hash function a second time. The output hash is 160 bits in length. First the key is mixed with the data, then the result is hashed, the hash is then mixed with the secret key again and the result hashed a second time. . SHA1 stands for Secure Hash Algorithm, and it's used to hash a value.

67 https://download.libsodium.org/libsodium/releases

		. This produces a hash of 160 bits length.
Encryption Protocol	AES/CBC/PKCS5Padding	. AES stands for Advance Encryption Standard, it's a general purpose block cipher standard that uses the Rijndael cipher. . CBC stands for Cipher Block Chaining. IT determines the AES encryption mode. This needs a random and unique Initialization vector (IV) per encryption request. . PKCS5Padding. This is used for padding, meaning that the size of the encrypted data is not always proportional to the size of the unencrypted data. This is a protection against reverse-engineering.
Key	Password value	This is the secret key used to feed the encryption algorithms. It needs to stay private between the producer and the consumer.
IV	Initialization Vector	The initialization vector must be random and unique per request but does not need to be kept private.

The encrypted data must follow the above specs, these should be supported in most programming languages as all the standards have been around for many years.

It's not the purpose of this work to dive much deeper into cryptography, but more information on best practices and the above protocols can be found in the OWASP website[68].

Since the encrypted data is saved in a relational database, the field sizes must be adapted to the encryption level, based on the protocol. For this and stability reasons, for now, there is only one protocol supported which is AES / CBC / PKCS5Padding.

Table 34: Field sizes non-encrypted vs encrypted (AES/CBC/PKCS5Padding)

Non-Encrypted	Encrypted
1 <= x <= 15	24
16 <= x <= 31	44
32 <= x <= 47	64
48 <= x <= 63	88
64 <= x <= 79	108

68 https://www.owasp.org/index.php/Cryptographic_Storage_Cheat_Sheet

80 <= x <= 95	128
96 <= x <= 111	152
112 <= x <= 127	172
128 <= x <= 143	192
145 <= x <= 159	216
160 <= x <= 174	236
175 <= x <= 191	256
192 <= x <= 207	280

With the information provided in this chapter two software providers should be able to implement this and be compatible, then they need to share a private key between them so that this platform doesn't have any knowledge or capabilities of decrypting the data.

There are higher level protocols that could be used, but with higher level protocols the space, memory and CPU resources used would also be higher, which could have a significantly impact on the system, so at this phase it's important to guaranty that the system will be able to maintain a high level of service, in the future other protocols can be supported (more on this subject see chapter).

3.3.7. Abuse Protection

To control abuse, several rate limiters were implemented, one for each service at the perimeter side, in this case, rate limiting is implemented in the router component, the implementation uses the module Play2 Guard[69] to facilitate it's implementation, it uses the token bucket approach (see chapter 2.3.5.1) and allows for the creation of IP and user based delimiters and IPs Black/White lists. It also supports direct integration with the platforms authentication and authorization library, silhouette, but since at the moment, the rate limiters implement are only IP based, these feature is not used.

The buckets are stored in general purpose in-memory collections, there are better solutions (see chapter), but this is an easy approach the meets the current requirements of the platform, it may be changed in the future if necessary. The buckets are stored and controlled by an Akka actor, there is exactly one actor per bucket, controlling the buckets with actors guaranties that the bucket can be used by concurrent requests and processes and relieves the implementation from race condition problems.

To better manage the memory resources occupied by the buckets, full buckets are removed, for example, if a bucket if not being used for some time, it doesn't make sense to keep that

69 https://github.com/sief/play-guard

bucket in memory, when a request arrives that's associated with that bucket then a new full bucket is created.

The rate limiters were injected in the platform as filters of Play's Framework, and they get it's settings from configuration files that can be changed on-demand (see chapter 3.10.2).

3.4. Segregation

Like previously stated (see chapter 2.3.4.3) a possible solution was to use Event-Sourcing with CQRS, however this is a more complex and time consuming solution to implement and because of this it was decided to implement a hybrid solution.

> "In particular CQRS should only be used on specific portions of a system (a Bounded Context in DDD lingo) and not the system as a whole. In this way of thinking, each Bounded Context needs its own decisions on how it should be modelled" (Martin Fowler, 2011)

Every micro-service has its own database, service has a thread pool of connections to it's database, meaning while there is threads available several SQL commands can run simultaneous, unless there is blocking in the database, so to prevent this blocking the most has possible each database is divided in two databases, one for reads and one for writes, additional scheduling the copy of the writes into the reads must also be done, the advantage of the scheduling is that it can be done in low traffic hours. One last thing needed is to separate the threads pools per database, Play's Framework facilitates this configuration by allowing an easy configuration of this separation of thread pools (see code 8).

```
database_one {
    dispatcher {
      executor = "thread-pool-executor"
      throughput = 1
      thread-pool-executor {
        fixed-pool-size = 9
      }
    }
}
```

Code 8 – Separating thread pools per database

The code above specifies an execution context that can be then appended to the repositories so that each access to the database uses this thread pool.

3.5. Use Case Overview

Following are two overviews, the first one represents the essential and common use cases this platform supports and the seconds represents the business oriented use cases.

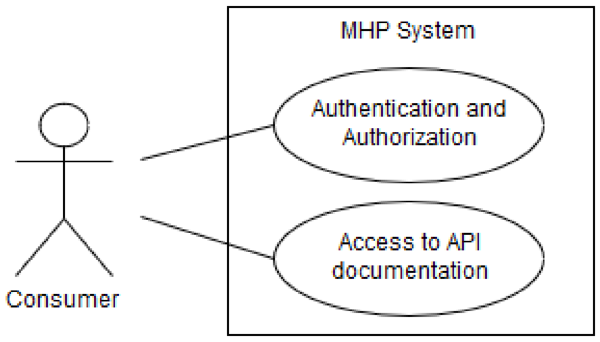


Figure 14: Common use cases diagram

Consumer: Represents an entity that consumes this platform APIs.

Authentication and Authorization: Security is mandatory in this platform so every request must first pass through authentication and authorization.

Access to API documentation: Consuming the API usually starts with requesting access and consulting the documentation.

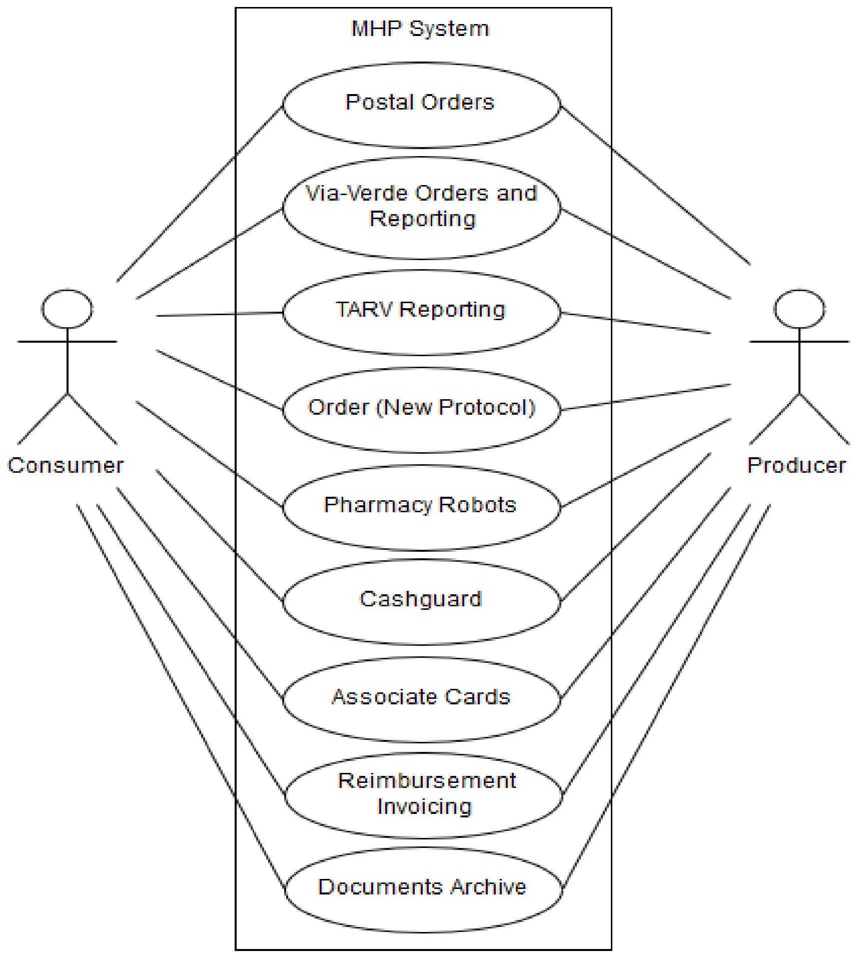


Figure 15: Business oriented use cases diagram

Producer: Represents a service provider.

Postal Orders: This a basic protocol that exists since 1995, it was designed specifically for the pharmacies and it allows them to order medication and other non-medication products from the suppliers.

Via-Verde Orders and Reporting: Via-Verde is a protocol developed by Infarmed, it as two objectives, allow the order of specific products and allow Infarmed to control the entire flow chain of those same products. The medication included in Via-Verde protocol is considered essential or vital in some cases, so Infarmed's control is justified so that it can better guaranty that those medications will be available to those who need it. For example, suppliers might, sometimes, when stock is low, refuse service to some clients to guaranty that other, considered better clients to that supplier, get their orders fulfil. With this protocol, Infarmed can check all orders from pharmacies to the suppliers and all orders from suppliers to the laboratories, thus controlling the flow chain, stock and availability of those medication.

TARV Reporting: This protocol has the same objectives has the previous one but for different medication, TARV or HAART stands for highly active antiretroviral therapy, and usually includes antiretroviral drugs to control HIV/AIDS infection.

Order (New Protocol): This a new developed protocol in this work, it supports the above order protocols, Postal, Via-Verde and TARV, and it allows for much more information to be exchanged in the order process, like campaign information or stock availability.

Pharmacy Robots: Not a lot of pharmacies have a dispenser robot, mainly because it takes a lot of space and it's very expensive, but those who do need their business software to communicate with the robot, to request products from the attendance balcony and insert received orders into the robot, the first one means that the user doesn't need to leave it's post because the requested medication is delivered next to him and the second means that the robot can automatically check all the medication that was received and report it back to the pharmacies business software, which in turn can then compare that with the original order.

Cashguard: This is a protocol to communicate with an automatic money safe, which usually it's at the users side in the attendance balcony, this safe, controls all insertions and retrievals of money, meaning that, with this protocol, the safe can be in contact with the business software and control the exact amount of what needs to go in and out.

Associate Cards: This a protocol to control pharmacy customers association cards, that typically give them medication discounts, for example if a card has lost it validity, this protocol can consult the responsible entity and report back to the pharmacy if the card is valid.

Reimbursement Invoicing: This is also a new protocol that allows the pharmacies to send invoices to the financial entities responsible for reimbursing the pharmacies for medication sold below its cost.

Documents Archive: At any giving moment, all entities that communicate with the MHP platform can request any document that belongs to them.

3.6. MHP Conceptual Architecture

From a general perspective the following diagram gives a clear view of the platform basic architecture and how everything is connected.

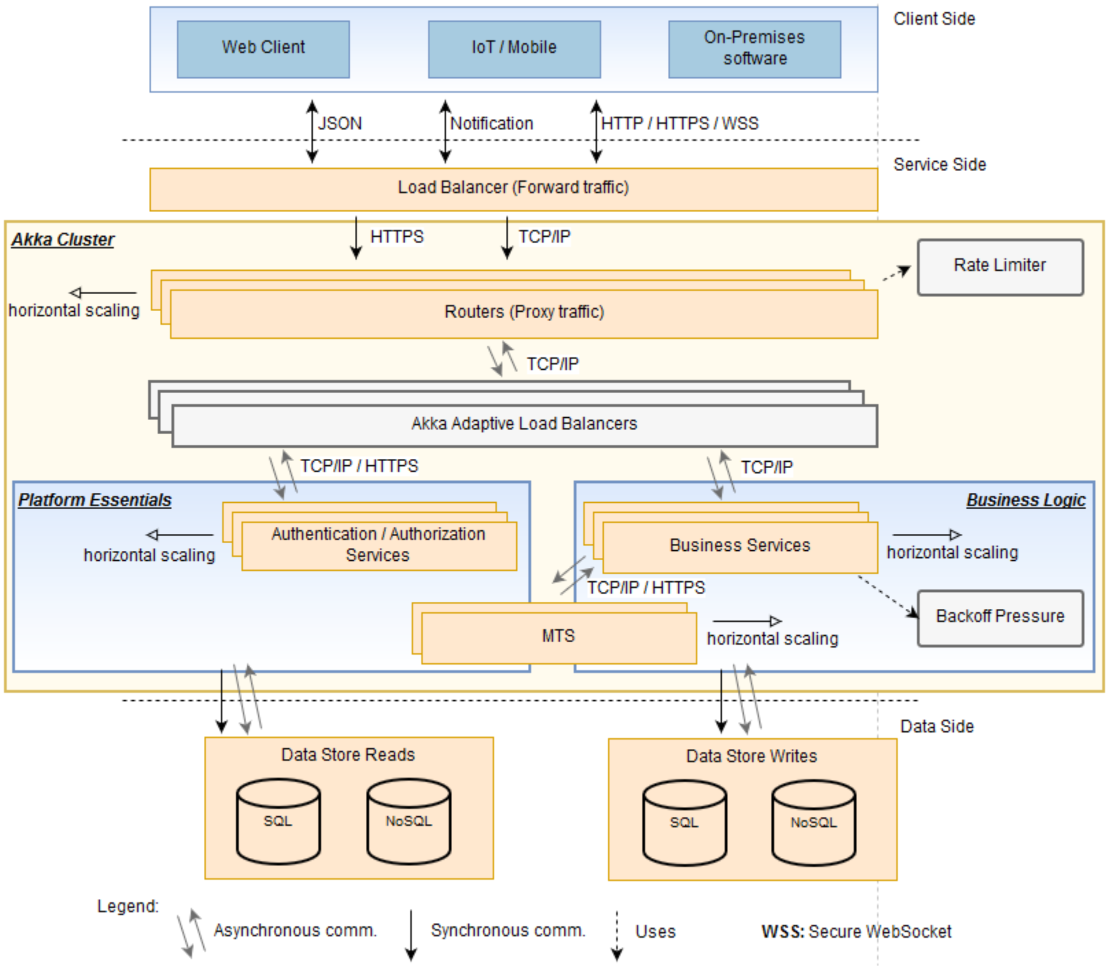


Figure 16: MHP Conceptual Architecture

The diagram above shows the platform divided in 3 layers, client side, service side and data side. The client side itself is not a part of the platform but it’s essential to understand how everything is connected. The platform itself is composed of the service side and the data side which are independent of one another. Following is a description of each layer.

3.6.1. Client side

This layer demonstrates how the clients communicate with the platform, communication is made through HTTPS and/or WSS (Secure WebSockets), there are no unsecured communication. If the client tries to communicate in an unsecured faction the load balancer redirects it to a secure channel (HTTP to HTTPS and WS to WSS). Message format is always in

JSON format, no other format is supported, there may be other message formats in the future, but there are no plans in supporting XML, mainly because it's a more complex and heavier format then the alternatives.

3.6.2. Service side

The service side is where the main components of this work are present. Technology wise, the components are all Play Framework instances inside the same Akka system, each instance has a predefined number of always running actors and can have any number of short lived actors created dynamically.

All instances are part of the same Akka cluster which gives them the capability of always knowing where the actors are, relieving the developer of such concerns. A couple of actors, can be one or more, are configured as Seed actors, these actors are then responsible to control and report the state of the cluster and of the actors. Any Akka instance can be a Seed actor.

This also allows for the horizontal growth of the entire system. Any instance can just be replicated and started in a different port than the existing instances, it will automatically be a part of the cluster.

3.6.3. Data Side

The data side of the platform is composed of relational databases (SQL) and non-relational (NoSQL) databases. Every service has its own database, but different business services can access the same database, for example, for pooling common data.

In some cases, data can be separated in a reads database and a writes database for improved performance and also higher complexity.

3.7. Components Overview

The following diagram represents the top-level components of the platform followed by a description of each of them. It represents the different kinds of components developed for this platform.

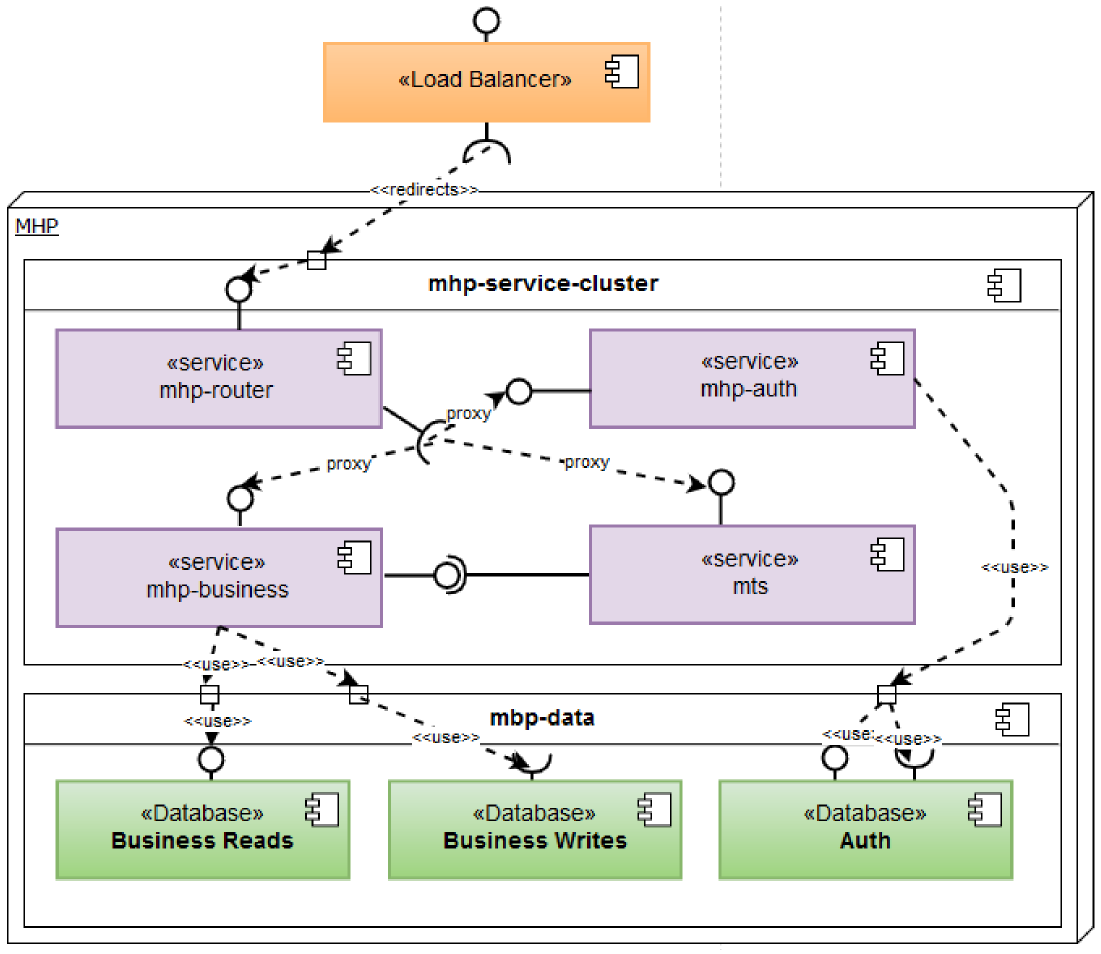


Figure 17: MHP Components Diagram

The following tables describes elements of the previous diagram.

Table 35: MHP Components Elements

Metric	Description
Load Balancer	Infrastructure provided load balancer to which all request goes through. This load balancer redirects the traffic to one of the *mhp-router* instances.
mhp-router	Interprets and validates all requests and load balances the same to the appropriate backend service.
mhp-auth	Responsible for authentication and authorization. The *mhp-router* uses this component to authorize all requests.
mhp-business	Represents all business service of this platform.
mhp-mts	Multi-Task Scheduler is responsible for processing all cron jobs.
db-reads	Relational database that supplies the *mhp-business* services with access to

	data.
db-writes	Relational database that supplies the *mhp-business* services with data storage.
db-auth	Non-Relation database that supplies the *mhp-auth* service with user persistence data (reads and writes).

3.8. Package Overview

All MHP apps use the same basic architecture approach for structuring the code. The system applies a basic MVC (Model View Controller) pattern but extends the same to Services, Repositories and Actors.

The following diagram represents the basic package structure of all MHP apps. Some services might have additional packages and complexity, but the fundamental of all apps is the same.

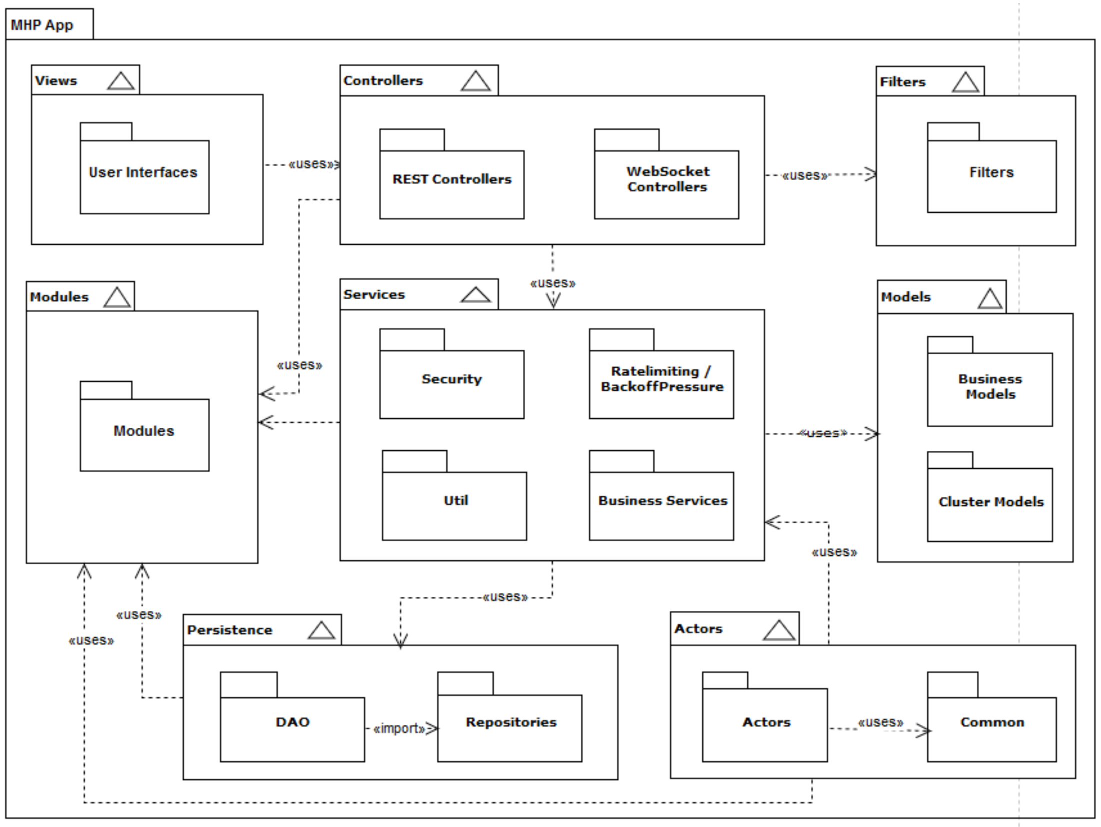


Figure 18: MHP Package Diagram

Table 36: MHP Package Elements

Metric	Description
Views: Represents the presentation layer. All instances have a basic UIs, but most of them don’t extend this functionality.	
User Interfaces	Contains the Web UI.
Controllers: Contains all controller’s classes. Controllers are access right after route validation. The Controllers make use of the Services to extend its functionality, abstract complexity and decouple for service logic implementation.	
Rest Controllers	Contains the controllers to service route RESTful requests.
WebSockets Controllers	Contains the controllers to service route WebSockets requests.
Filters: All controllers Action functions validate the request using the Filters.	
Filters	Contains platform specific Filters and custom Filters.
Models: Contains all business logic classes. The classes are used by the Services.	
Business Models	Contains the service related business logic classes.
Cluster Modes	Contains the business logic classes that are specific to cluster related needs, like, monitoring and reporting.
Persistence: Contains persistence related classes to access and persist data. The Persistence layer is used by the Services.	
DAO	Contains the data access objects classes.
Repositories	Contains the repositories which use the DAOs.
Services: Services are basically what join everything together and is where the most complex logic is implemented, this Services can also extend other services depending on the complexity of the app in question.	
Security	Contains hashing and encryption services.
RateLimiting / BackoffPressure	Contains abuse control services.
Util	Basic, cross-cutting concern utilities. Can help alleviate the complexity of the other Services.
Business Services	Contains all business-related apps.
Actors: Actors are a mix of business classes with service classes. They have a special place in the platform because they are used for most concurrent code which has specific requirements, like keeping its assets private, this is because is the only way to protect against racing conditions.	

Actors	Contains the actors
Common	Contains common resources that can be used for other actors in other systems. For a class to be sent to another Actor in another system, the class must exist in both systems and the FQN (Full Qualified Name) of that class must be a match.
Modules: Modules is where it’s possible to extend Plays Functionality and where Dependency Injection configuration is defined.	
Modules	Contains one or more module classes that defines all functionality that can be extended though dependency injection.

To make changes inside this architecture, it’s easier if there's a clear defined path from which to approach and start producing code.

The following diagram gives a mental, ordered and systematic approach that a developer should take to change or add business related functionality to a service/app.

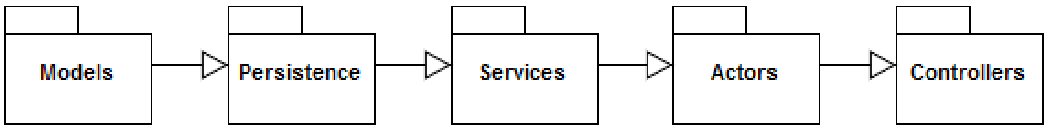


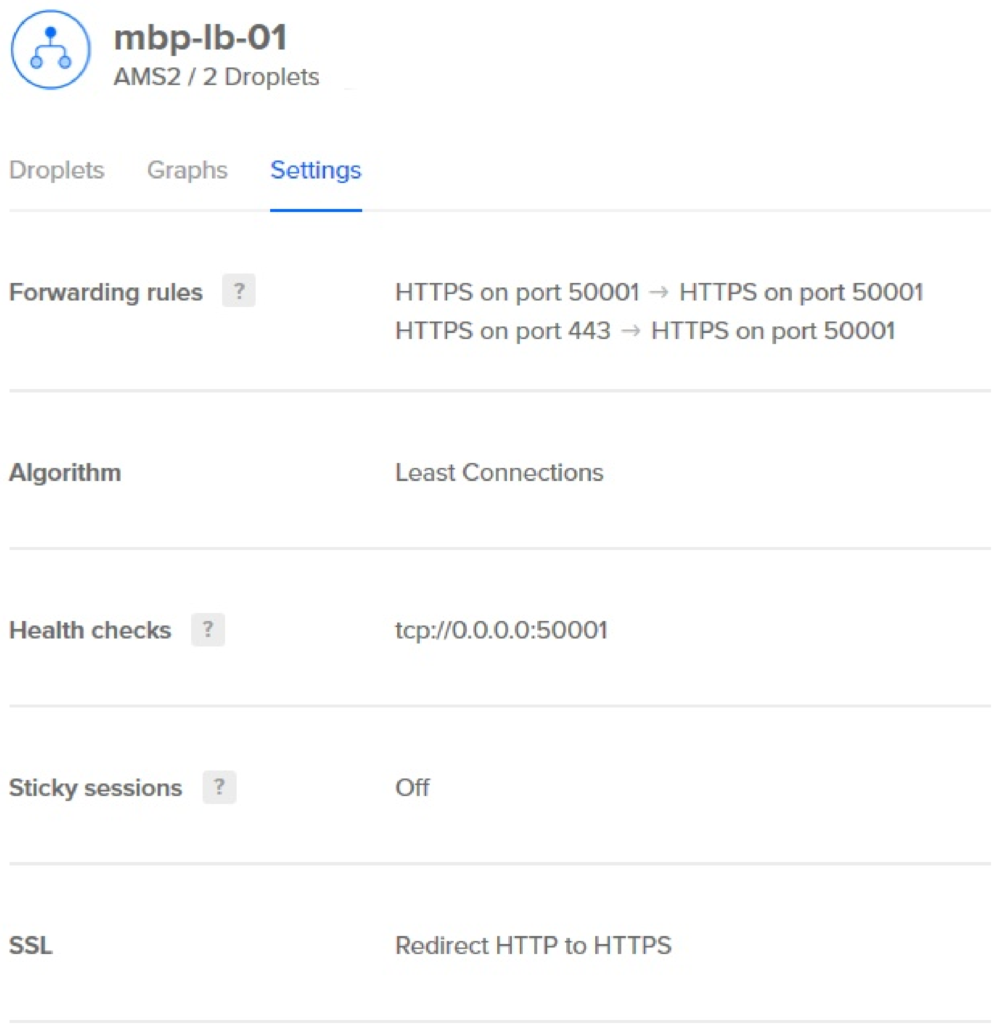
Figure 19: MHP Package Development Workflow Diagram

3.9. Load Balancing

3.9.1. Perimeter Load Balancer

All traffic goes through the main load balancer which is then forwarder to the Akka router components. The load balancer is set up to forward specific traffic, which only includes HTTP. HTTPS, WS and WSS, redirecting the non-secure connections to secure ones. It uses a pass-through TLS validation meaning that the certificate validation is not the responsibility of the load balancer but the Akka routers. It also uses TCP/IP but not through it’s forward mechanisms, TCP/IP is only used to ping the Akka routers as to control its health state, if a specific instance is not considered healthy then it doesn't forward traffic to that instance.

The load balancer is not an Akka cluster member, and it’s a component provided by the infrastructure provider. All major cloud infrastructure providers support load-balancing capabilities, although it’s features might vary slightly. The features described here are standard for all load balancers. The following figure shows an example configuration of this load balancer in the provider DigitalOcean.


mbp-lb-01

AMS2 / 2 Droplets

Forwarding rules ?	HTTPS on port 50001 → HTTPS on port 50001 HTTPS on port 443 → HTTPS on port 50001
Algorithm	Least Connections
Health checks ?	tcp://0.0.0.0:50001
Sticky sessions ?	Off
SSL	Redirect HTTP to HTTPS

Figure 20: DigitalOcean Load Balancer Configuration

Note the algorithm for load balancing, least connections means that the traffic will be forwarded to the instance with the least connection, the only available alternative, in this case, is round-robin which send the traffic to the instances in a sequential order. Also in the following rules is setup the TLS pass-through, the alternative would be to add the certificates to the load balancer.

3.9.2. Akka Adaptive Load Balancers

Traffic is load balanced from the routers to the services through Akka actors, this uses light weight TCP/IP communication. The load balancing patterns can take several formats, probably the most common format is the Round-Robin router, which blindly sends traffic to one service after the other, this is not ideal because the number of requests sent to an instance does not represent the resources status of that instance, for example, a request can be more CPU intensive than another, take more time to completion, etc., because of this, the platform uses an adaptive load balancer that takes advantage of the cluster metrics to know at each time which is the instance with the most resources available and sends the traffic to that instance (see next figure).

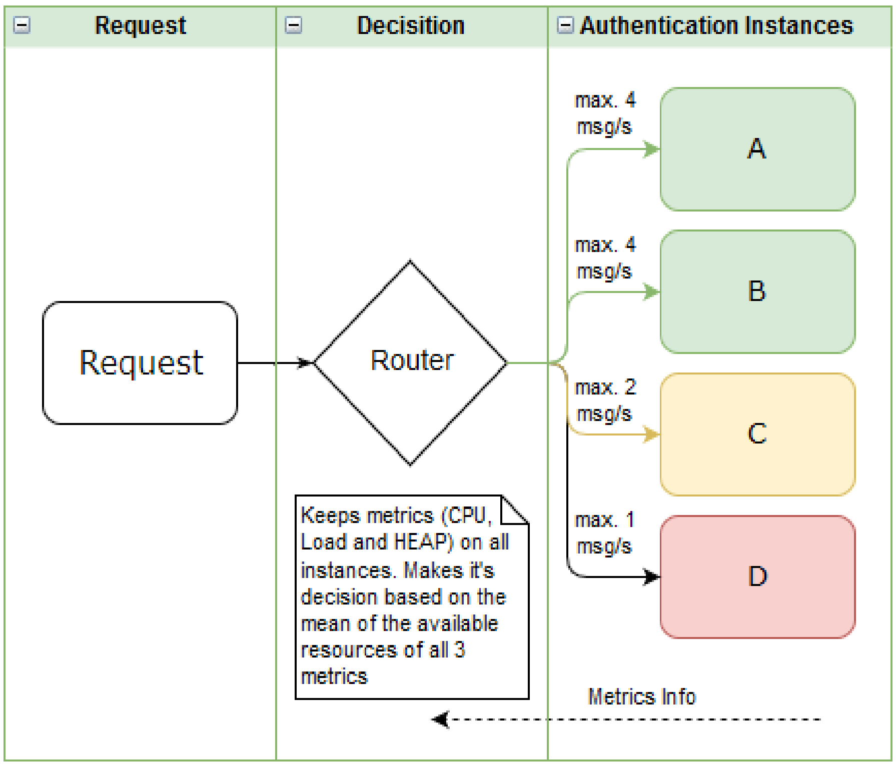

Request
Decisition
Authentication Instances
Request
Router
Keeps metrics (CPU, Load and HEAP) on all instances. Makes it's decision based on the mean of the available resources of all 3 metrics
max. 4 msg/s
A
max. 4 msg/s
B
max. 2 msg/s
C
max. 1 msg/s
D
Metrics Info

Figure 21: Akka load balancing

To implement this load balancer the following, simple, configuration must be used:

```
akka {
  actor {
    provider = "cluster"
    deployment {
      ## Auth Router
      /routerActorSupervisor/router/authRouter {
        router = cluster-metrics-adaptive-group
        # metrics-selector = heap
        # metrics-selector = load
        # metrics-selector = cpu
        metrics-selector = mix

        routees.paths = ["/user/authWorker"]

        cluster {
          enabled = on
          use-role = auth
          allow-local-routees = off
        }
      }
    }
  }
```

Code 9 – Example configuration of a load balancing router

The router "*cluster-metrics-adaptive-group*" is the implementation provided by Akka that uses the cluster metrics to validate the instances resources, these instances are identified by the "*routees.path*", which hold the actor designated name, and every instance that belongs to the same cluster, has the role "*use-role = auth*" and has an actor with that specific name will be part of the load balancing. The cluster metrics[70] this router uses are CPU, Heap and Load (see Table 37), but can be configured to use just one of them.

It's possible to monitor the metrics inside an actor that subscribes to cluster metrics events, this is done inside the *WatcherActor* which has the responsibility of monitoring the current instance, each instance has a *WatcherActor.* The monitoring of the metrics can be disable by configuration, typically this is not needed in production. The metrics logged can be as follow:

Table 37: Cluster Metrics (see footnote 70)

Metric	Description
HEAP	Used and max JVM heap memory.
Load	System load average for the past 1 minute. This value can be found in Linux systems and the more closest to the number of cpu/cores the bigger the bigger the load. Weights based on: 1 - (load / processors).
CPU	CPU utilization in percentage, sum of User + Sys + Nice + Wait.
Mix	Uses a combination of the previous three. It's based on the mean of the remaining the resources off al three metrics.

Table 38: Cluster Metrics Monitoring Example

Metric	Example
CPU	Address: akka.tcp://msb@127.0.0.1:41001, Timestamp: 1507204507449, SystemLoadAverage: 15%, CpuCombined: 10%, CpuStolen: 2%, Processors: 4
HEAP	Address: akka.tcp://msb@127.0.0.1:41001, Timestamp: 1507204507449, Used: 205.2136058807373 Mb, Committed: 267386880, Max: Some(477626368)

3.10. Akka routers

Like the perimeter load balancer, all traffic must go through the Akka routers, these routers also, function as Load Balancers, API Gateway and Reverse Proxy. Since they are a part of the Akka cluster they can be scaled horizontally. The perimeter Load Balancer sends traffic to the

70 https://doc.akka.io/docs/akka/2.5/scala/cluster-metrics.html

router instance that has the least active connections and the router itself also uses load balancing to send traffic to the backend services, however, the router uses a smarted Load balancer that sends traffic to the service instance that has the most resources available. The following activity diagram shows the steps that each request must go through.

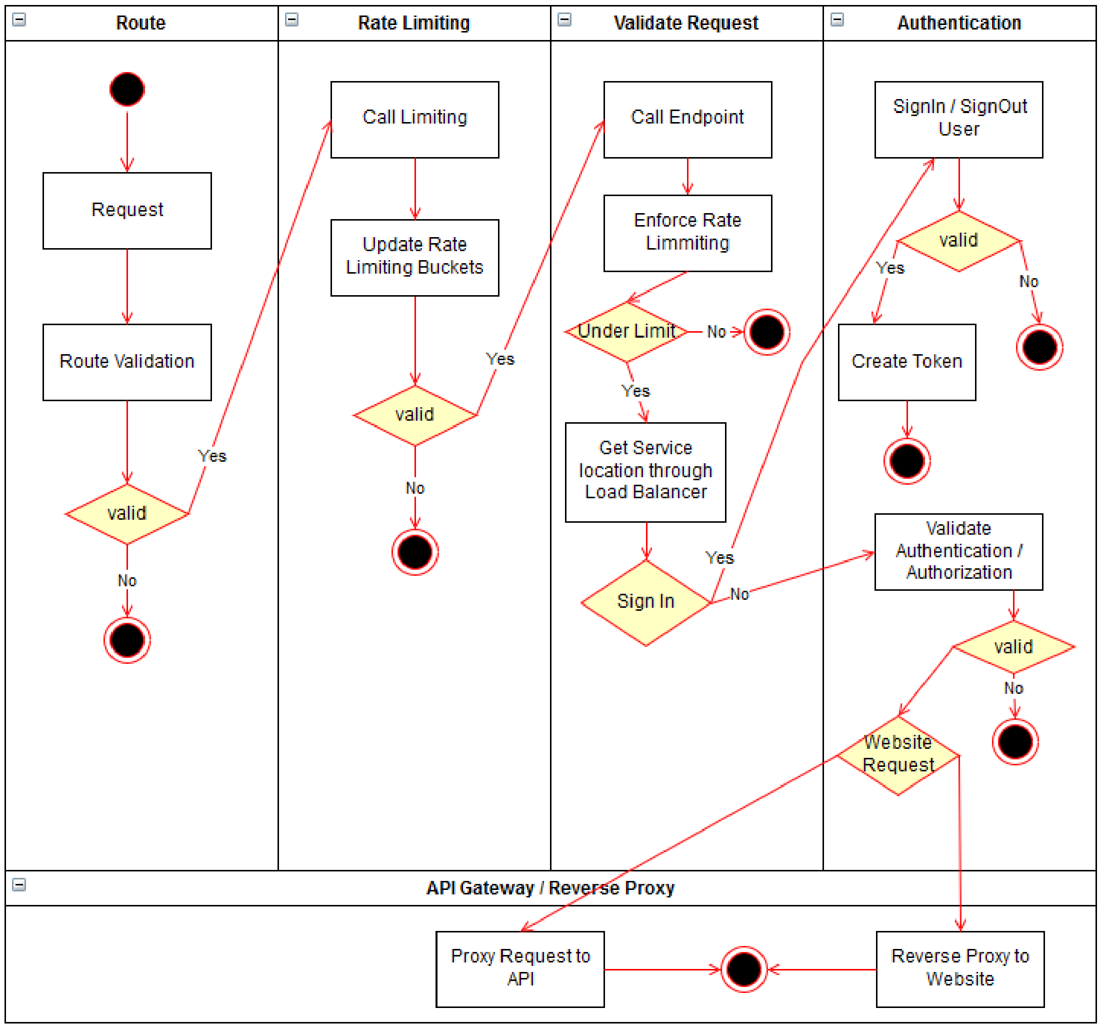


Figure 22: Router component activity diagram

3.10.1. Route Validation

If the route is not recognized then the request / connection is discarded, otherwise it's goes through the defined Play filters, which are (in order):

Table 39: Router Filters

Order	Filter	Description
1	Logging Filter	This is the first filter because it logs all requests to the platform.
2	Guard Filter	The second filter is the rate limiter validation. This filter only saves the rate limiting state because the rate limiting imposition may be subject to other rules that can only be enforced later (see below).
3	Allowed Hosts Filter	This filter allows specific hosts to bypass the next two filters.
4	CSFR Filter	This filter protects against CSFR attacks.
5	Security Headers Filter	This filter validates that predefined mandatory security headers are Ok. Currently this filter is not active in the router.

3.10.2. Rate Limiter and Request Validation

Rate limiting allows the routers to control abuse from clients. The rate limiting can be configured differently for each service, this makes it necessary to only enforce the rate limiting when the requests reaches its route Action[71]. Every request must go through the following rules:

71 Action is the Play Framework approach to the function that is called that corresponds to the route requested.

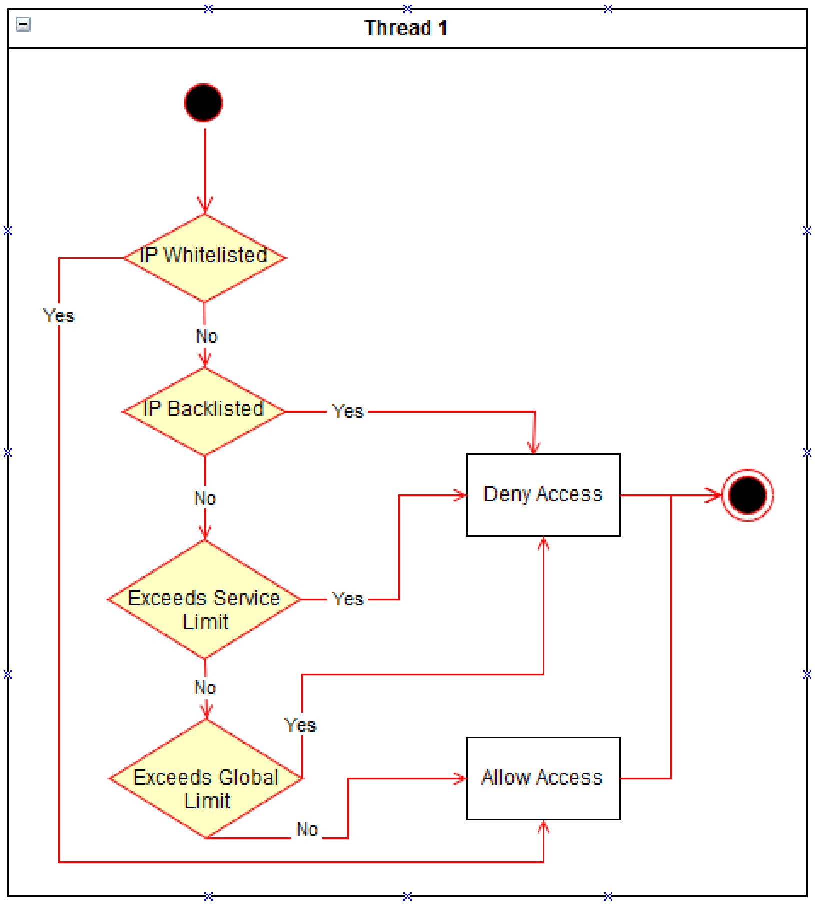


Figure 23: Rate Limiter activity diagram

Configuration of the rate limits are in a configuration file so that it can be changed at any moment, or even deactivated if needed, the configuration is done in two parts, essential configuration that is part of the framework PlayGuard and this Platform specific configuration. This configuration includes the global limits, for all services, and the limits for just specific services, in this example the authorization service, it also shows another king of limit, that is based on the number of HTTP errors, also per IP. An example of such configuration can be seen in Annex .

To add the rate limiting to a specific action the easiest way is to concatenate the rate limiting action to the routes Action, like in the following example:

```
def proxy(service: String, resource: String, id: Option[String]) =
  (Action
    andThen rateLimitServ.getKeyRateFilter(service)
    andThen rateLimitService.getHttpErrorRateFilter(service)).async {
implicit req =>
…
…
}
```

Code 10 – Example configuration of a load balancing router

Notice the "*andThen*" after the Action, what happens is that the request goes through all actions, each of them with its validations and rules, the "*rateLimitServ.getKeyRateFilter*" applies the general rate limits per IP and the "*rateLimitServ.getHttpErrorRateFilter*" applies the rate limits per IP and HTTP errors. "*rateLimitServ*" it the service implementation that integrates the PlayGuard library with this platform implementation and configuration.

If the request isn't blocked by rate limiting, it's validated for its destination, for this a request to the load balancer is sent that returns the instance address (see chapter), after that the operation is validated, if it's a *SignIn* operation than it goes directly to its destination service, if not, then an *"is authenticated"* message is sent to the authorization service and only then, if it's authenticated successfully, is it sent to its destination service.

The authorization steps can involve two scenarios, if it's a *SignIn* and if it's successfully then it creates and returns a valid JWT to be used in future requests, if it's an "is authenticated" operation than it validates the received JWT for its authenticity and, if valid, it validates the users authorization to the service requested and returns its result. In each case the HTTP of the request is sent to the authorization services for validation.

3.10.3. Gateway and Reverse Proxy

The user requests can be redirected to two router components, if the request is for a website then the request goes through the Reverse Proxy and if the request is to a backend service API then it goes through the API Gateway. The reason for this is that to proxy requests to websites the router must be transparent to the user, meaning that all data that compose the user request must be forwarded to the backend website transparently and in the case of an API request, the router does not need to worry about every detail of the users data, because it might not be relevant for the services API. This setup also means that the API Gateway can be more lightweight than the Reverse Proxy.

The following table shows some advantages and disadvantages with this approach.

Table 40: Pros and Cons of using an API Gateway and Reverse Proxy

Pros	Cons
Single entry point Load Balancing between services Limited surface attack More control over all API requests Can force only secure connections for all services	Compromising the router can compromise the system Services are more insecure from the inside Changes to the router can affect the entire platform

3.10.3.1. API Gateway

All API requests to backend services go through the *APIGatewayCtrl* controller which in turn uses the *ProxyService* to proxy requests. This development takes advantages of the Play Framework provided libraries to work with all related Requests and Responses concerns. The following tables shows the necessary steps to implement this process and the code can be seen in Annex . The first table details the process of the APIGatewayCtrl, which us just a set of validations and decisions and the second table shows the actual proxy process of the ProxyService.

Table 41: API Gateway code description

Step	Description
Take the standard validations	All requests go through the standard Filters.
Get service virtual server if exists	Because a backend service can only accept HTTPS requests, the proxy needs to take advantage of the SSL Certificate function called SNI (Server Name Identification) which takes effect by adding a virtual server Identification that basically identifies the certificate recognized domain name. If a backend service is on HTTP then the virtual server is ignored.
Find out if service use HTTPS	These steps identifies the connection type. If the service uses both HTTPS and HTTP than HTTPS is preferred.
Validate Authorization	Go through the authorization process.
Ask router for backend address	There can be several instances of the backend service, so it's necessary to ask the router actor to which address to send the request, since the router actor can take advantage of the load

	balancing.
Proxy requests to backend	Sends the request to the backend.
Process Response from backend and Reply to client	Processes the response. This validates if the response is acceptable or an error before returning the same to the client. If it is an error than it validates the error and returns an appropriate response.

In the next table is the actual Request structure that needs to be sent to the backend service. The code can be seen in Annex .

Table 42: API Gateway proxy code description

Step	Description
Construct the URL	Constructs the URL based on the initial request.
Add requests headers	All headers from the initial request are sent to the backend service, with the exceptions of Content-Type and Content-Length because this need to be added through the HTTP Entity[72] and should not go directly into the headers.
Adds the virtual server	For certificate validation
Adds the query string	If the initial requests had query strings this are also added, unchanged, to the request.
Add the body to the request	The body is also added to the request.
Adds the HTTP method	The same HTTP method has the initial request is also added only if the method doesn't come as a parameter, in this case the method is changed
Send the Request	The requests are sent to the backend service, at this point, since all communication is done asynchronously, the method returns to the API Gateway controller that called this Service.

3.10.3.2. Reverse Proxy

The Reserve proxy is a little more complicated than the API gateway, because of two things, first it needs to address more parts of the Requests and second because it's basically unknown

72 https://www.w3.org/Protocols/rfc2616/rfc2616-sec7.html

what the traffic will be the best approach is to stream the results, contrary to downloading all content and only after process the response. The Play frameworks library in conjunction with Akka streams helps in implement this streams approach.

Requests that are not part of the of the API Requests are redirected to the Reverse Proxy controller. The following example shows some route definitions in Play's Framework routes file, the first few routes (Auth Routes) are redirected to the API Gateway and the second ones to the Reverse Proxy.

```
# Auth Routes (API Gateway Routes)
GET   /api/auth   controllers.APIGatewayCtrl.proxy(service: String ?= "auth",
resource: String ?= "", id: Option[String] ?= None)
GET   /api/auth/*resource   controllers.APIGatewayCtrl.proxy(service: String ?=
"auth", resource, id: Option[String] ?= None)
POST   /api/auth/*resource   controllers.APIGatewayCtrl.proxy(service: String ?=
"auth", resource, id: Option[String] ?= None)
PUT   /api/auth/*resource   controllers.APIGatewayCtrl.proxy(service: String ?=
"auth", resource, id: Option[String] ?= None)
DELETE   /api/auth/*resource   controllers.APIGatewayCtrl.proxy(service: String ?=
"auth", resource, id: Option[String] ?= None)

# Reverse Proxy Routes
# Note: this routes should be the last
GET  /  controllers.ReverseProxyCtrl.proxy(service: String ?= "", resource:
String ?= "")
GET  /*resource  controllers.ReverseProxyCtrl.proxy(service: String ?= "",
resource)
POST  /*resource  controllers.ReverseProxyCtrl.proxy(service: String ?= "",
resource)
```

Code 11 – Router routes example

The routes file is interpreted by Scala's pattern matching, because of this is important that all API routes be defined first to guaranty that only the ones that are not a match with the API routes are redirected to the Reverse Proxy, for example, a POST to *api/auth/signIn* would match the 3rd route, a GET to *some_other_resource would match the 7th route, and a call to* www.some-site.com/ would it the 6th route.

The reverse proxy if a more complicated process than the API Gateway code, so it's divided in two functions, *proxy* from the *ReverseProxyCtrl* and *relayRequest* from the *ProxyService.* The first function does basically the same thing has the *APIGatewayCtrl*, it determines where to send the request and processes the response, and the second function prepares and sends the request itself. The code can be seen in Annex .

The next table describes the steps of the code in the second function.

Table 43: Reverse Proxy code description

Step	Description
Construct the URL and add to request	Constructs the URL based on the initial request
Adds original request method	Same method has the original request always.
Add virtual host	For certificate validation
Add request headers	Same headers as original request
Add request query string	Same query string as original request
Add cookies	Add all cookies from original request
Add body	Add body of original request
Stream the response	If the response contains the Content-Length it's possible to stream the complete response, if not, the request must be must be downloaded using the chunked transfer encoding, which is only supported for HTTP 1.1 clients, if the client is an HTTP 1.0 client, a 505 error is returned. Chunked encoding[73] allows the server to send a response where the content length is not known, or for potentially infinite streams, while still allowing the connection to be kept alive and reused for the next request. The response headers and cookies must be returned to the client.

3.10.4. Router Dependencies

Besides the essential dependencies the router only needs one additional dependencies:

- "com.digitaltangible" %% "play-guard" % "2.1.0"

This dependency, Play Guard, is the library used for the Rate Limiting concerns (abuse control).

73 https://github.com/playframework/playframework/blob/master/framework/src/play/src/main/scala/play/api/mvc/Results.scala

3.11. Authorization Services

Authentication and authorization concerns are an independent component in this platform. This has the advantage of separating these concerns from the rest of the services that can then focused more on its business requirements and if anything needs to change in the authorization process, only these components need to be updated.

3.11.1. Authorization Dependencies

Following is a list of all authorization specific dependencies:

Table 44: Authorization specific dependencies

Dependency	Description
"org.reactivemongo" % "play2-reactivemongo_2.12" % "0.12.6-play26"	Reactive (asynchronous) MongoDB driver.
"com.mohiva" %% "play-silhouette" % silhouetteVer, "com.mohiva" %% "play-silhouette-password-bcrypt" % silhouetteVer, "com.mohiva" %% "play-silhouette-crypto-jca" % silhouetteVer, "com.mohiva" %% "play-silhouette-persistence" % silhouetteVer, "com.mohiva" %% "play-silhouette-testkit" % silhouetteVer % "test", "com.iheart" %% "ficus" % "1.4.1"	Silhouette libraries, to deal with authentication and authorization, based on cookies and JWT. The ficus library if a special configuration library used by silhouette.
"org.webjars" % "requirejs" % "2.3.1", "com.adrianhurt" %% "play-bootstrap" % "1.2-P26-B3"	Libraries for needed for the web UI.
Swagger 2.0	Swagger is not added to the service through SBT, but rather as a public resource. This is because there need to be made some changes to its library, like changing the company logo, etc. And this is only possible by changing the source code directly, and if it was added through SBT it would be overwritten every time the app was compiled. Swagger is needed for the

	documentation platform.

3.11.2. Access to API Documentation

Because this is a commercial project, the API documentation of the business services can't just be open to everyone, so, to protect access to this resource while still maintaining the API documentation online, an online Portal was created so that only authorized personal can access. The online portal uses the Swagger framework to display the documentation, basically the documentation is specified in YAML files, that the Swagger framework can interpret and display a page with the documentation in a friendly and useful format.

Following are example of this implementation showing the authorization API. Confidential information has a black square over it.

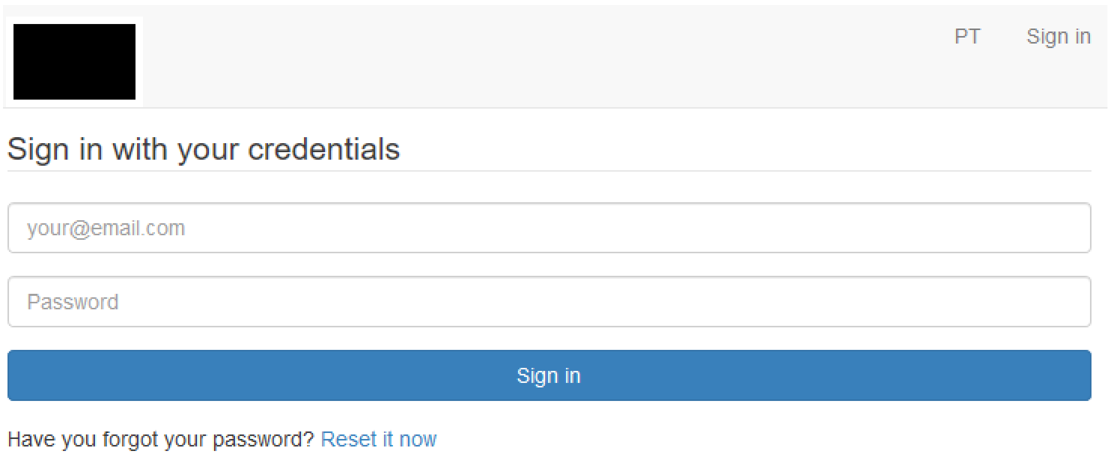
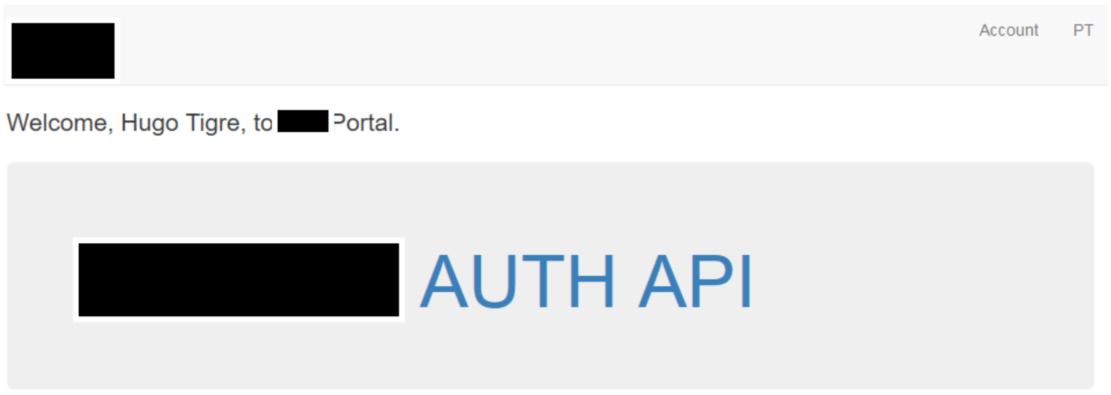

[redacted]
PT
Sign in
Sign in with your credentials
your@email.com
Password
Sign in
Have you forgot your password? Reset it now

Figure 24: MHP documentation portal - signIn

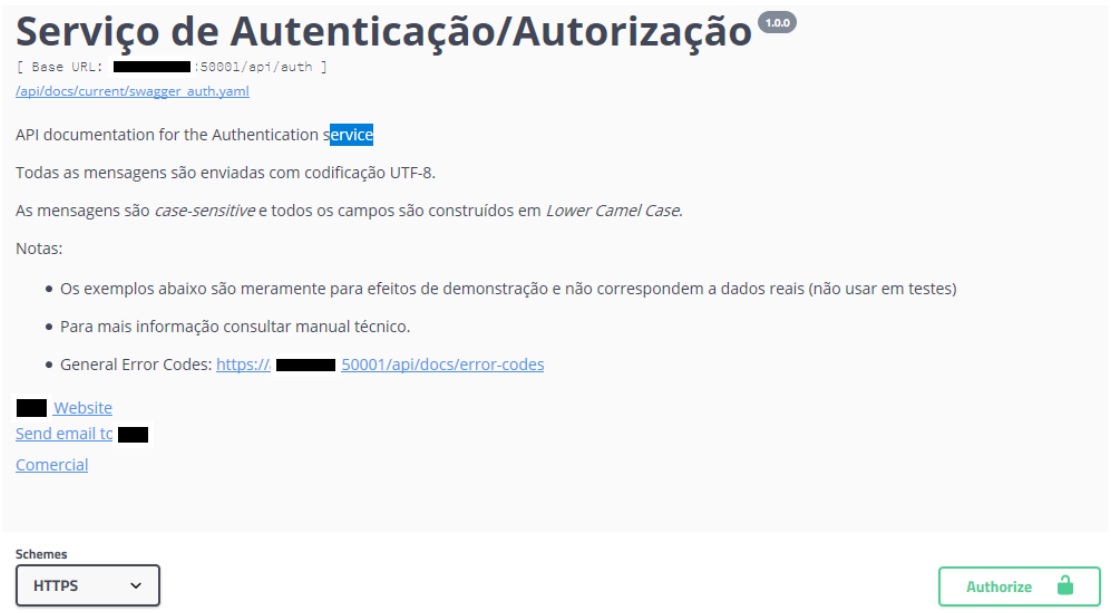
Credentials need to be provided before users can access the documentation. After the user logs in it is redirected to a page with the documentation available to him, meaning that some users might have access to some documentation that others don't, this is controller through the authorization process.



Welcome, Hugo Tigre, to [redacted] Portal.

[redacted] AUTH API

Figure 25: MHP documentation portal – Link to docs


Serviço de Autenticação/Autorização 1.0.0

[Base URL: [redacted] :50001/api/auth]

/api/docs/current/swagger_auth.yaml

API documentation for the Authentication service

Todas as mensagens são enviadas com codificação UTF-8.

As mensagens são *case-sensitive* e todos os campos são construídos em *Lower Camel Case*.

Notas:

- Os exemplos abaixo são meramente para efeitos de demonstração e não correspondem a dados reais (não usar em testes)
- Para mais informação consultar manual técnico.
- General Error Codes: https:// [redacted] 50001/api/docs/error-codes

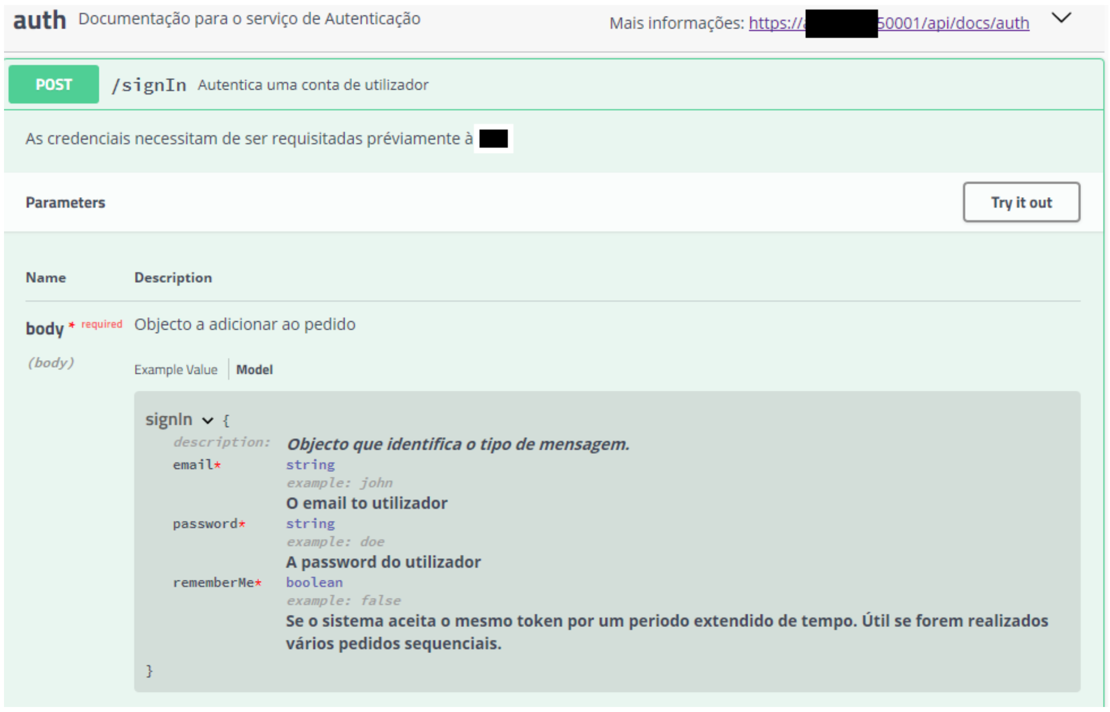
[redacted] Website

Send email to [redacted]

Comercial

Figure 26: MHP documentation portal – Authorization docs header

All API documentation have an information header that has basic, but important information about the API, like in the example above, message coding, case formatting, link to a page with the available error codes, etc. Note that the information above is in Portuguese because, for now, the platform is only available for the Portuguese market.


auth Documentação para o serviço de Autenticação

Mais informações: https://[redacted]50001/api/docs/auth

POST /signIn Autentica uma conta de utilizador

As credenciais necessitam de ser requisitadas préviamente à [redacted]

Parameters

Try it out

Name	Description
body * required (body)	Objecto a adicionar ao pedido

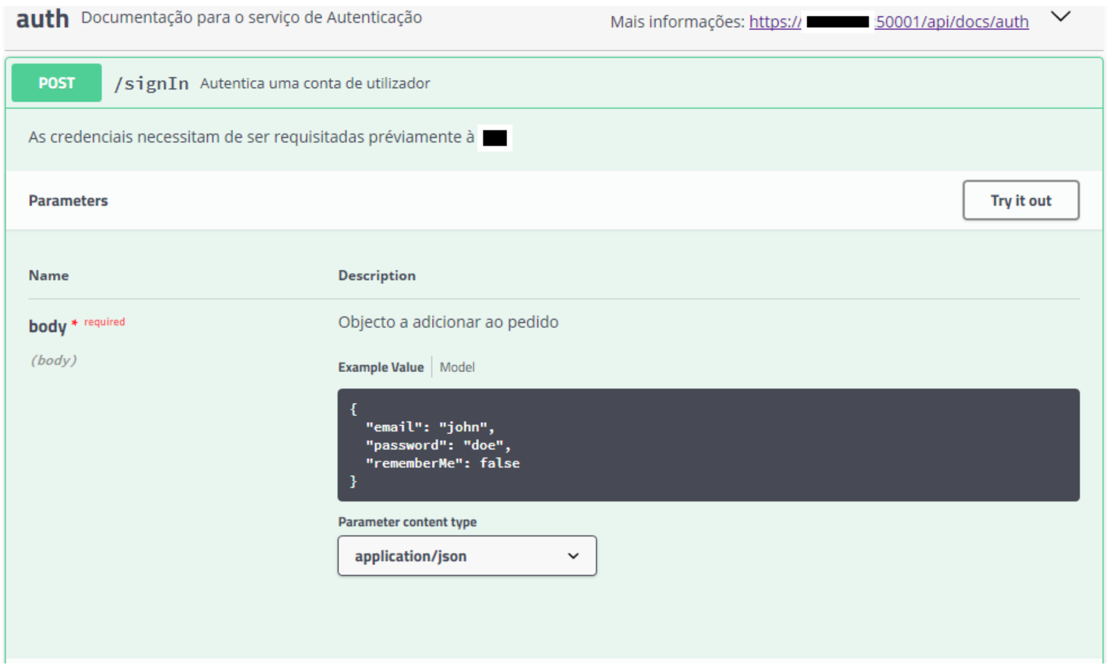
Example Value | Model

```
signIn {
    description:  Objecto que identifica o tipo de mensagem.
    email*        string
                  example: john
                  O email to utilizador
    password*     string
                  example: doe
                  A password do utilizador
    rememberMe*   boolean
                  example: false
                  Se o sistema aceita o mesmo token por um periodo extendido de tempo. Útil se forem realizados
                  vários pedidos sequenciais.
}
```

Figure 27: MHP documentation portal – Authorization docs model

After the header information is a list of all supported messages and the necessary information to be able to make a successful request. In the figure above, it's possible to see the *signIn* message model with each fields description and constraints.

On the figure below, it's the same section but instead of showing the model, it shows an example for easy viewing only, it is not real data, has it described in the documentation header (see Figure 26).


auth Documentação para o serviço de Autenticação

Mais informações: https://[redacted] 50001/api/docs/auth

POST /signIn Autentica uma conta de utilizador

As credenciais necessitam de ser requisitadas préviamente à [redacted]

Parameters

Name	Description
body * required (body)	Objecto a adicionar ao pedido

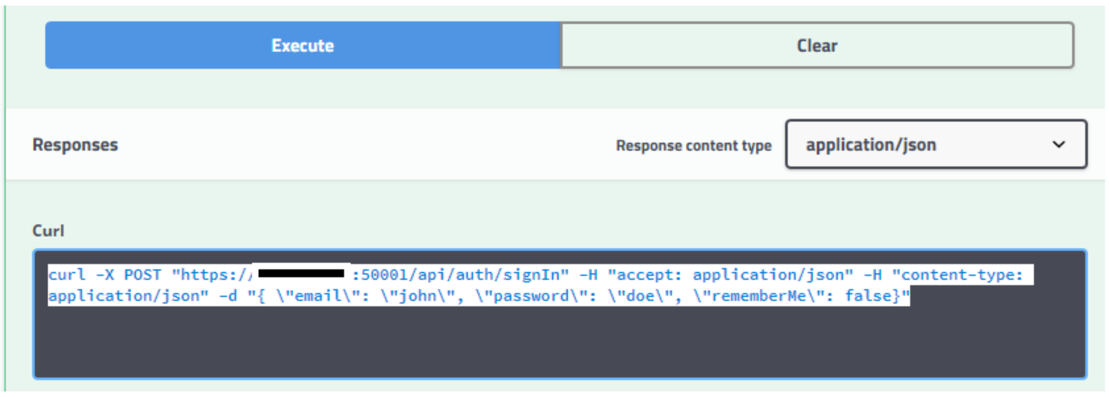
Example Value | Model

```
{
  "email": "john",
  "password": "doe",
  "rememberMe": false
}
```

Parameter content type: application/json

Figure 28: MHP documentation portal – Authorization docs example

The documentation pages also allow the user to test the API calls, by clicking in the "Try it out" button It's possible to edit the example and using a curl command, that's filled out automatically, and pressing the "execute" button it's possible to send the message to the API Gateway (see next figure).



Responses

Response content type: application/json

Curl

```
curl -X POST "https://[redacted]:50001/api/auth/signIn" -H "accept: application/json" -H "content-type: application/json" -d "{ \"email\": \"john\", \"password\": \"doe\", \"rememberMe\": false}"
```

Figure 29: MHP documentation portal – Authorization docs try out

An example of a documentation file can be seen in Annex 6.6.

3.12. Cluster and Main System Configuration

Several important features of the platform can be configured or extended with configuration files, this chapter demonstrates the most relevant features to work in coherence with an Akka cluster.

The following table show the main configuration parameters that are essential to the platform and a complete configuration file can be seen in Annex .

Table 45: Plays Framework with Akka cluster main configuration file parameters

Parameter	Example Value	Description
play.http.secret	asoifxxxxx9658745q038dyhgqcas	This key must be kept secret has it is used by the framework for security reasons, like for example to encrypt cookies.
play.server.http.port	disable	Disables HTTP port.
play.server.https.port	50001	Specifies the HTTPS port.
play.ws.ssl.trustManager.stores	[{path: $java.home}/lib/security/cacerts, password: "changeit"}]	Specifies the Java trust stores, this is needed of the services connects to other HTTPS services.
play.filters.hosts.allowed	[“.”]	Inside the cluster, access should be restricted to only the necessary services, their IP addresses can be configured in this setting.
akka.actor.deployment	(see annex 6.5)	In this section it’s possible to configure load balancing routers among others.
akka.cluster.seed-nodes	["akka.tcp://msb@127.0.0.1:41001","akka.tcp://msb@127.0.0.1:41011"]	From within the cluster, one or more micro-service must be configured as a seed. Seed nodes are responsible from sharing cluster member information between other nodes.
akka.remote.nett	${app.address}	Identifies the node’s IP address.

y.port.hostname		
akka.remote.nett y.port.port	${app.akka.port}	Identifies the node's IP port. Note that when a service is replicated in the cluster, this value and the previous one must be different so not to have a conflict between the cluster.

3.13. Multi-Task Scheduler

The MTS service is basically responsible for two purposes, executing jobs on-demand and scheduled jobs.

Jobs must be able to:

1. **Be scheduled:** they should be able to run at a specific time;
2. **Be configured:** job configuration should be done via configuration files, i.e., no development should be needed;
3. **Be logged:** job activity should be registered (start, stop, timestamps, errors, etc...);
4. **Be executed on command:** some jobs are only useful if they can be executed on demand and not scheduled;
5. **Be cancelled:** for example, long running jobs;
6. **Be prioritized:** high priority jobs should have priority over low priority ones, but the implementation should not allow the low priority ones for never running;
7. **Be pausable:** not a main feature, but it can be useful to pause a running job and resume it at a latter time;
8. **Validate dependencies:** some jobs have dependencies, can be other jobs or specific resource, and should not be executed when these are not available;
9. **Be securely aware:** jobs should be able to validate authentication and authorization;

Most requirements can be met with available resources provided by the available frameworks, with the exception of the Cron jobs. Akka has functionality which they call a scheduler, but it has limitation, the Akka scheduler is an easy and simple way to run code at specific intervals, for example, running code every 5 minutes, but it's no possible to schedule, for example, to run everyday at 5:00pm. To overcome this limitation the library *akka-quartz-scheduler*[74] is used. It also uses Akka actors, but it allows for more flexible definition of Cron

74 https://github.com/enragedginger/akka-quartz-scheduler

Jobs. The Cron specification follows the Quartz Cron Expression[75] Language that match the standard Unix cron syntax.

Jobs are defined in a configuration file like the following example:

```
akka {
  quartz {
    defaultTimezone = "UTC"

    schedules {
      Every30Seconds {
        description = "A cron job that fires off every 30 seconds"
        expression = "*/30 * 19,8-18 ? * *"
        timezone = "Europe/Lisbon"
        calendar = "CronOnlyBusinessHours"
      }
    }

    calendars {
      CronOnlyBusinessHours {
        type = Cron
        excludeExpression = "* * 0-1,22-23 ? * *"
        timezone = "Europe/Lisbon"
      }
    }
  }
}
```

Code 12 – Configuration of a Cron job

The above example shows just a simple example, in the *schedules* section it's possible to define any number of schedules and on the calendars section it's possible to define different king of calendars that the schedules use. The expression setting can take any value compatible with Quartz Cron Expression. For more examples see footnote 74 and 75.

The execution of the schedules is also easy, it just needs an actor and the instantiation of that actor through the *akka-quartz-scheduler* library, for example:

```
val scheduler = QuartzSchedulerExtension(as)
val myActor = actorSystem.actorOf(MyActor.props)
scheduler.schedule(name="Every30Seconds", receiver=myActor, msg="some message")
```

Code 13 – Connect an actor to a schedule

75 http://www.quartz-scheduler.org/api/2.1.7/org/quartz/CronExpression.html

3.13.1. MTS Dependencies

Following is a list of all MTS specific dependencies:

Table 46: MTS specific dependencies

Dependency	Description
"com.typesafe.play" %% "anorm" % "2.5.3,	Anorm library, for running commands in the database
"org.postgresql" % "postgresql" % "42.1.4"	Postgresql driver
"com.enragedginger" %% "akka-quartz-scheduler" % "1.6.1-akka-2.5.x"	Quartz-Scheduler library for defining Cron jobs.

3.14. Business Services

Business Services are any instance that implements the business requirements of the platform. It's not in the purpose of this work to detail every aspect of the business architecture, so this chapter address the more relevant characteristics and technology concerns of the Business Services.

3.14.1. Dictionary Data

Dictionary data is all data that can and is shared between the stakeholders, for example, product names, product metadata, etc., to the point that makes sense to uniform the same data between the stakeholders. To accomplish this, a database to contain dictionary data was created, which is a combination of information provided by Infarmed and custom added information which is produced by keeping a close contact with the correct entities which can provide relevant data and keeping it daily updated. Updating the database on a daily basis is very important because this kind of data changes every day, and some data, like price medication is crucial to some stakeholders, like the pharmacies.

The update of this data is the responsibility of the MTS component, and is a combinations of web-crawling to gather data and SQL scripts to update the database. There is also some manual updates in some case, mainly fixing erogenous data and adding or changing data that can only be obtained by interpreting legislation or protocols.

3.14.2. Medication Subsidies

When a patient has a prescription from its doctor, part, or all, of the cost of the medication can be subsided by the state or a private entity. The way this is calculated, is in real time in the pharmacy, meaning it's the pharmacies responsibility to know how this works, of course, it would be very difficult and impractical to have the pharmacists calculate this by hand, so it's the pharmacy business software that provides this functionality. The problem here, is that information must be always up to date and the rules to calculate the fees can be different for each medication. Each entity has its own rules, that can change at any time, these changes, sometimes, imply not only updates to the data, but also to the software, and, because some changes are only notified after they were published, it's very difficult to keep up to date. This creates a very complex system to manage and requires specialized people that can interpret the rules and regulation that dictate the subsidised values.

This platform helps with this process in two ways, by keeping update to date information on these rules and regulation, and for providing the calculations for the subsidised values, for this, a data structured specifically designed for this purpose is necessary and the mathematical operations or formulas must be implemented in code, the variables of these operations are influenced by the data. The following diagram shows the main data structure implementation for this, additional fields are removed from the diagram for clarity.

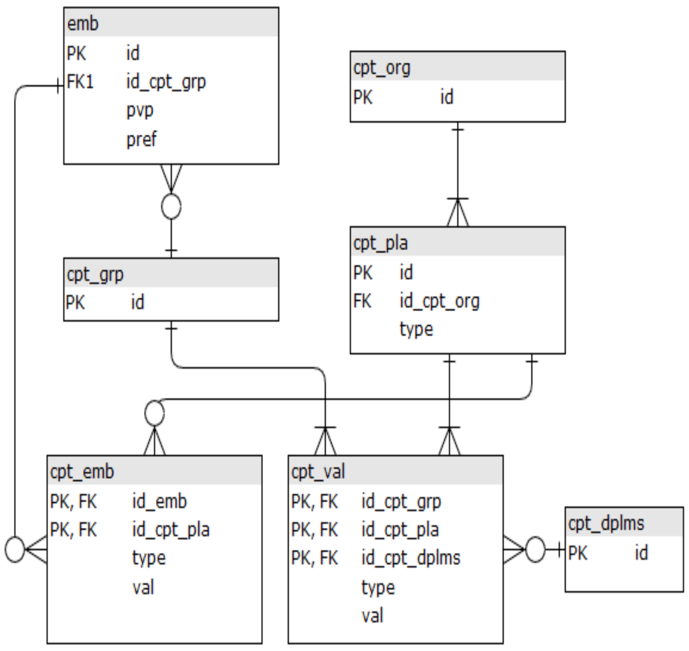


Figure 30: MHP medication subsidies relational diagram

- **emb:** This is the medication table, each medication has a unique id, some have a fixed price, meaning it always has to be sold at that price and a reference price, which is used in some subsidies calculation, and a reference to the *cpt_grp* table (see below);
- **cpt_org:** This table has all the entities that subsidise the medication;
- **cpt_pla:** This table represents the plan of the subsidies, it's what points to the rules and regulation, one entity can have different plans;

- **cpt_grp:** This table allows each medication to belong to a group, this makes it possible for the same medication to be subsided by different plans and rules;
- **cpt_val:** This table has the values that can be subsided, the type of value and the notion of a diploma, which is part of a legislation. For example, if an organization has a plan that subsides 20 of a medication, that 20, can be a percentage over the *pvp*, a percentage over the *pref*, the exact value that is subsided or the exact value that the client pays. This can all be changed if a diploma comes into the equation, meaning that a diploma can change the values of the same plan, this is typically the case, but not exclusively, of severe diseases, like Alzheimer's, Cancer, etc., which, if the patient has that disease, he can pay less for the medication or even have the medication for free. This table also connects with *cpt_grp* so that it can have different values for different groups of products.
- **cpt_dplms:** This table as the list of all diplomas;
- **cpt_emb:** This table is for special cases where it's impossible to have rules, in some cases they don't exist, and the only way to know how to subside a medication is to have it' value directly defined. In this case the medication does not relate to the plans by belonging to a group but rather directly to the plan.

This structure holds the data necessary to calculate the subsided values and it's the starting point to understand how the system works, but it still depends on the formulas that are implemented in code, which by itself is a very big subject and is not included in this work.

The biggest problem with this system is that many changes must be done manually, because they are delivered in the form of legislation and not in a compatible digital format that could be interpreted by an algorithm, also, it's impossible to predict the changes because each entity can have its own rules.

3.14.3. Backpressure Strategy

Backpressure strategies are applied where streaming occurs, in this platform case this is where WebSockets are implement, so, at the moment, backpressure strategy is on only available for web-socket connection. WebSockets are implement using the Akka framework which provides the following strategies:

Table 47: Platform Essential Dependencies

Strategy	Description
Backpressured	When new connections arrive it backpressures the upstream publisher until space becomes available. This means that the client connection might see the server connection as slower or not responsive. Play's Framework also tries to indicate to the client that the connection is alive but busy (see chapter 3.1.6). This type of strategy might seem the

	obvious choice, but it makes it more difficult to predict every client behaviour.
Drop Buffer	When new elements arrive, it drops the entire buffer to give space for new elements.
Drop Head	Drops the oldest element in the buffer to give space to the new elements.
Drop Tail	Drops the newest element in the buffer to give space for the arriving element.
Drop New	If the buffer is full then the new element is dropped.
Fail	If the buffer is full it completes the stream with a failure.

This platform uses the Drop New strategy, in favor of the Backpressured strategy, because it's an easier to control and predict client behavior, i.e., if there is no space for new requests than this requests are simply dropped. In the future it might prove more efficiently to turn to the Backpressured strategy, but this will need significantly more testing.

3.14.3. Support for Legacy Architectures

In some cases, there is a need to integrate old software, like for example software that just runs on Windows but at the same time the same software is running on servers which clients can access through remote access technologies like, terminal services, remote desktop, etc...

In this cases communicating with local devices can be tricky, it possible to transfer data through a remote desktop connection but it's nothing stable, it has different behaviors in different version of the operating system, and is very limited, some knows hacks include passing keystrokes and catching mouse movement. So, to keep local devices connected to remote, cloud based installations this platform implements the following architecture:

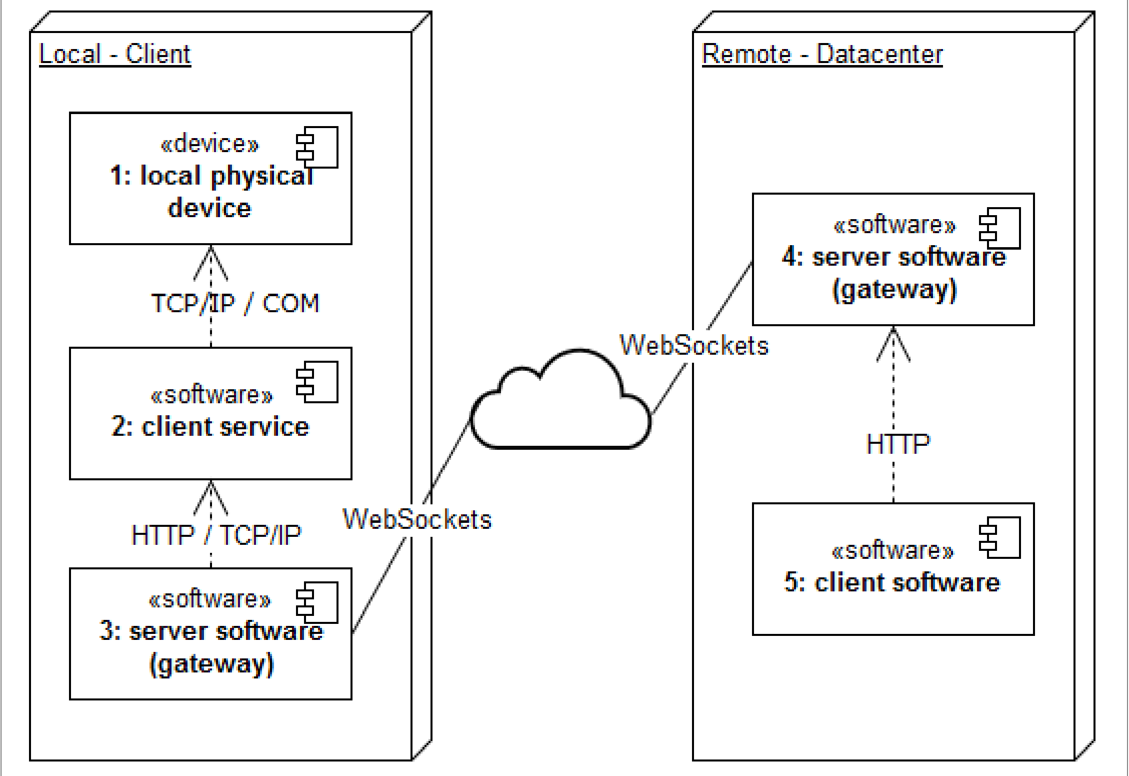


Figure 31: MHP legacy architecture solution

The previous diagram represents the following workflow:

1. Component 3 starts and connects to component 4. At this time it registers with a specific id that can map the component with the client location;

   1.1. Component 4 needs to save state of connected clients;

   1.2. Component 3 needs to provide two important features:

      1.2.1. Be always connected, for the server to be able to push notification, for this it uses WebSockets connection;

      1.2.2. Reconnect if connection is lost.

2. Component 5 needs to communicate with component 1, so it sends a message to component 4 which knows where component 1 is located because of the connection between component 3 and 4;
3. Component 4 relays the message to component 3 with component 2 ID;
4. Component 3, knows how to reach component 2, and relays the message to it;
5. Component 2 finally sends the message to component 1;
6. Component 2 receives a response from component 1 and sends the message back until it reaches component 5.

Some practical examples for this scenario:

1. Getting readings of card devices;
2. Communicating with a TPA[76];
3. Communicating with a robot.

Because communicating between A and B involves 5 components, it's important to have control over the logs so that troubleshooting is possible without having to access each component one at the time. This is accomplished by reporting the errors through email and also by tagging an application identification in the status messages that identify where it originated, this tag is encrypted so that the application identification becomes private, like in the following example:

```
{
  "status": {
    "i": 200,
    "s": "Ok",
    "v": "1.0.0",
    "ai":
"fe27e42dabbe531002ae1e75e6b5eabc5f482e7cc3e122a7ef717ccb20929fd6a64edbf00a2ae505
22578b49"
  }
}
```

Code 14 – Status message example

76 Terminal de Pagamento Automático (Credit Card POS Terminal)

3.15. Deployment

The platforms architecture allows for easy horizontal growth, which in turn also facilitates the deployment requirements. The following diagram shows the deployment of this platform components into the datacenter provider.

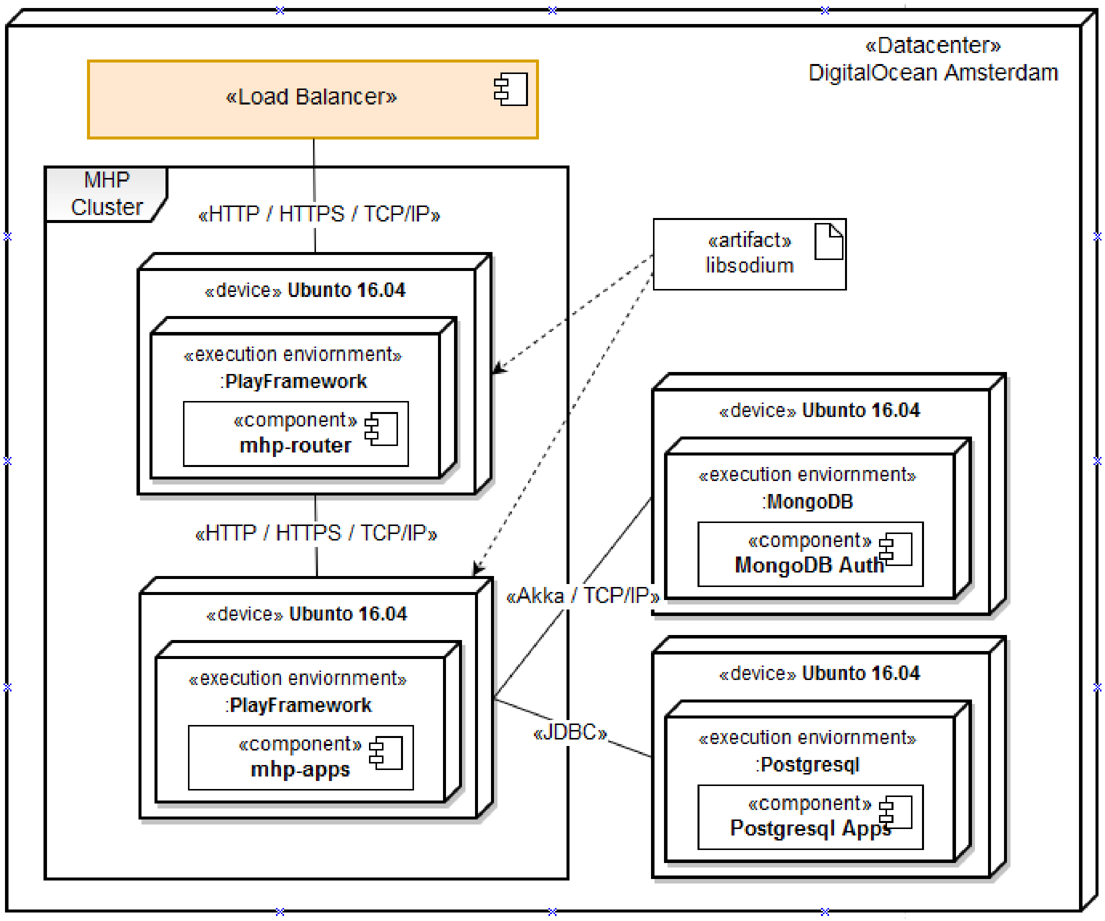


Figure 32: MHP Deployment diagram

Following are the most important characteristics of the previous diagram:

- All components are sent to Digital Oceans datacenter;
- All virtual machines have Ubuntu 16.04 OS;
- The deployment of the router and apps is basically the same, every instance is deployed to a different virtual machine;

- The same applies to the databases, which have different deployment procedures but, basically, follow the same rules, which is one per virtual machine;
- The artefact *libsodium* is the only external artefact that needs to be installed in all virtual machines that have an instance of an app.

The next diagram shows a more detailed networking view.

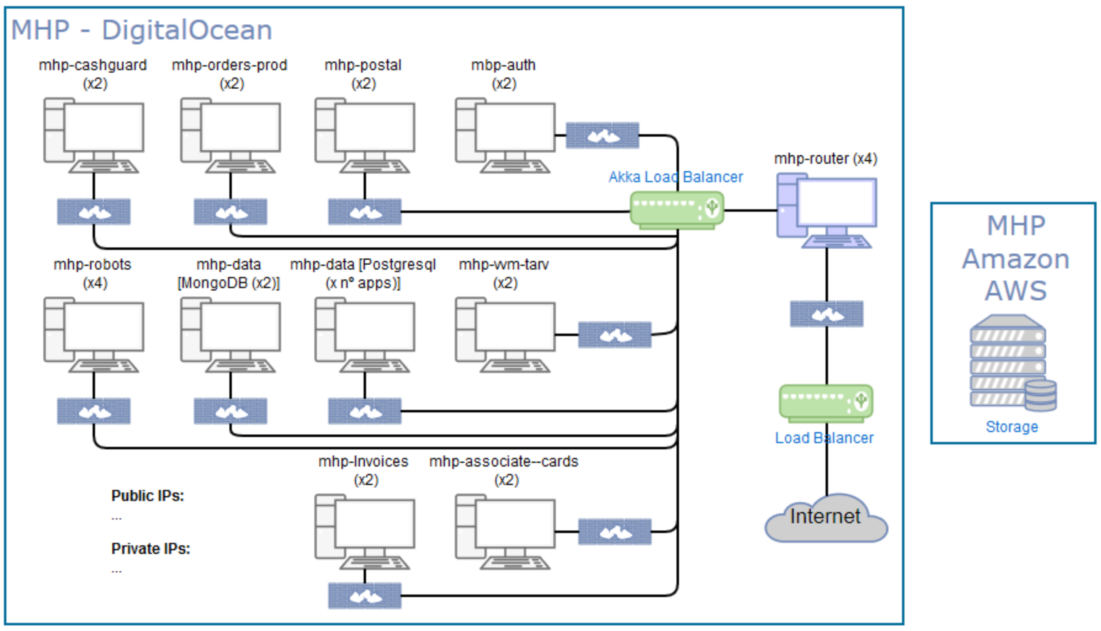


Figure 33: MHP network diagram

From the previous diagram is important to note the following:

- Every virtual machine has an independent firewall, this is not installed in the OS, it's a separate component provided by DigitalOcean;
- Every service/app is deployed at least 2 times, some more, depending on the resource requirements;
- Backups are saved in Amazon AWS;
- Outside load balancing is provided by DigitalOcean while inside load balancing if provided by the router components;
- The public and private addresses have been removed for security reasons.

3.16. Performance Testing

For performance testing purposes a HTTP load testing tool called Vegeta[77] was used, this tool was chosen among others[78] because it allowed testing the framework in a useful and easy way, by having a big range of features and reporting capabilities embedded into the tool.

The process can be resumed to the following:

1. Create a file with the message body (ex: body.json)
2. Create a file with the endpoints called (ex: targets.txt)
3. Execute the Vegeta tool:

Table 48: Vegeta command line arguments

Argument	Example	Description
-cpus	2	nº of CPUs
attack	-	Specifies the type of test (attack / report). Se example below.
-rate	10	requests per second
-duration	30s	
-targets	targets.txt	Specifies the endpoint to test
-body	body.json	Specifies the body of the request. If there are several different requests than the body can be separated in the same file by two carriage returns.
-header	"Content-Type: application/json"	Specifies the Content-Type

77 https://github.com/tsenart/vegeta
78 For example, the tool Apache Benchmark (ab) doesn't allow testing with certificates that it doesn't recognize, which is a big limitation in a testing tool.

These commands can be concatenated with the reporting features to run the tests and produce the results, like in the following example:

```
vegeta.exe -cpus 2 attack -rate=50 -duration=30s -targets=targets.txt
-body=signin.json -header="Content-Type: application/json" | vegeta report
-reporter=plot -output=report.html
```

Code 15 – Vegeta command line example with attack and reporting

The above command produces the following result:

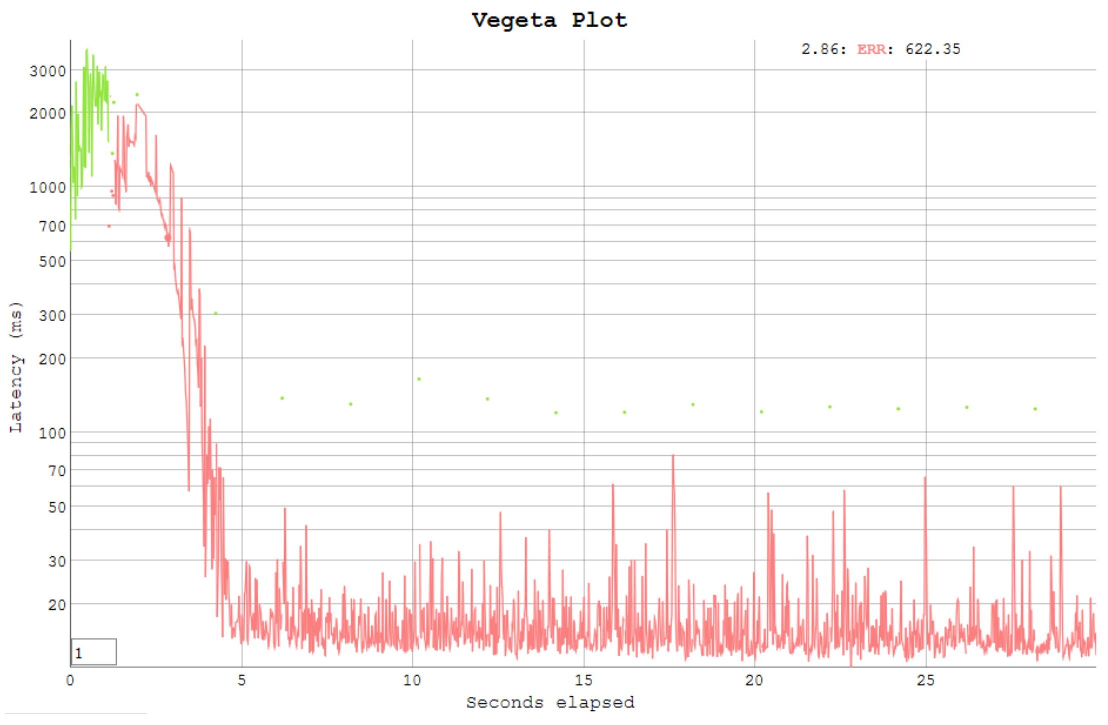


Figure 34: Performance testing with rate limiting

From this graph is possible to reach the following conclusions:

1. By doing 50 requests per second to the *singIn* resource, the rate limiting in the API Gateway starts denying traffic at around 2 seconds;
2. The response comes back a lot faster since the request is not being processed;
3. The little green dots, means that a request, occasionally, gets through, this is because the rate limiting is gradual, and in this case new tokens are available one every 2 seconds (see chapter).

Now, it the same request is made without the rate-limiting the following graph is produced:

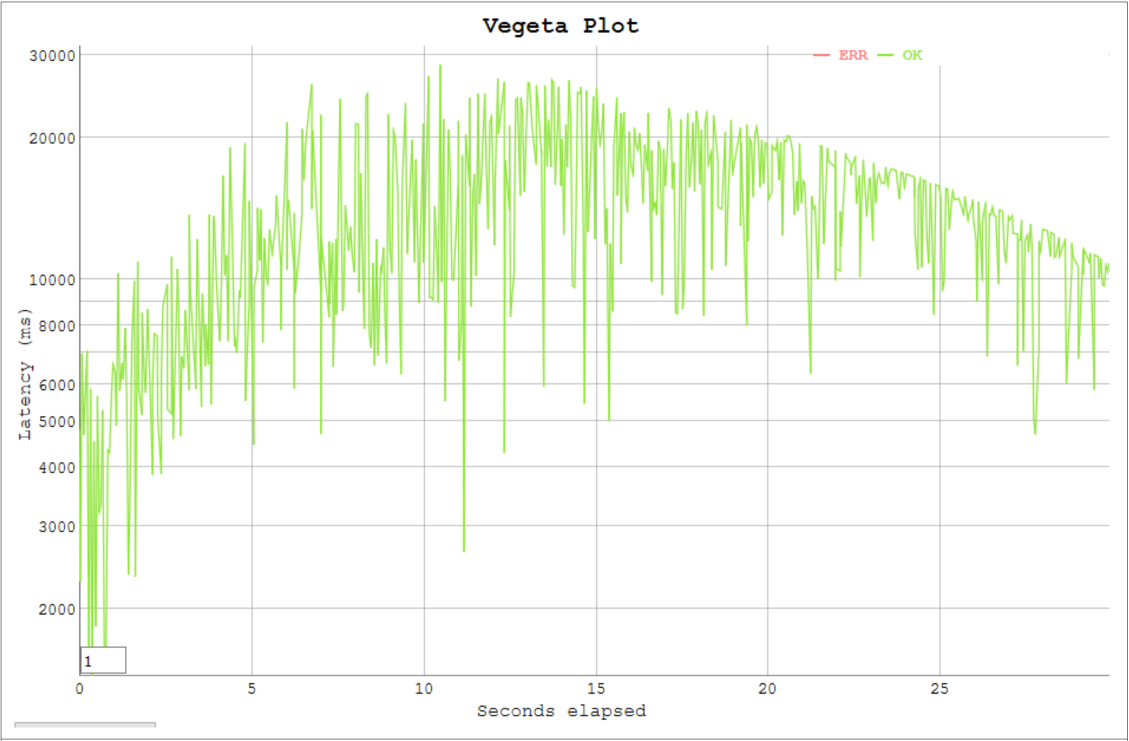


Figure 35: Performance testing without rate limiting

From the graph above the conclusions are:

1. Because there was no rate limiting the all requests were successfully;
2. The time it took for each request increased significantly because the platform could not keep up with so many requests at a time, and so, at least one request took 30 seconds to complete, which is not acceptable;
3. This shows not only the limitation of the platform, in terms of requests per second per kind of request, but also, that without the rate limiting, one client can decrease another client perception value of the platform.

In conclusion, despite the 429 HTTP error codes that the first scenario produced, it is the best scenario and the preferred approach.

4. Conclusions

Most choices of architecture and technologies where influenced by a mix of personal experience, documentation, proven use-cases and testing, which gives a certain level of assurance in that choices, however for the future evolution of this platform there must exist more metrics and benchmarks of different solutions to assure even more that changes to the platform are the right ones.

4.1. Accomplished Objectives

This work considers that the main objectives have been fulfilled, leaving just a few needed improvements for future work. Below is the appreciation of this work objectives.

a) The proposed security concerns have been addressed, there are still improvements to be made, but overall, security is assured by the only allowing encrypted communications, protecting resources with authentication and authorization mechanisms and allowing for the encryption of sensitive data;

   Clients only pay for what they really use and there are no entry fies, so small companies can still use the service without impacting too much on their financial budget;

b) Protocol support is basically accomplished, not all proposed services were implemented, only one of the pharmacy robots was implemented, mainly because, business priorities changed during development, TPA support was added despite not being in the initial objectives and all infrastructure requirements where implemented, also, missing business requirements will inherit from the existing implementations and architecture, so the work is already laid out;

c) All the business services implemented act has a proxy for other services, so data transformation was added to each one;

d) End-to-End Encryption is offered in a form of technical specification, the entities using the platform still need to implement it on their side. This process must be improved significantly;

e) The dictionary database provides the unification of business data, but it needs continuous attention to keep the data up to date.

f) Abuse protection keeps system resources available for all clients, maintaining equal service quality for all;

g) At the moment all data is saved and not deleted, keeping a permanent history. Depending on future resource consumption and financial stability this requirement should be reviewed;

h) The platform components can scale horizontally without the need for additional development, and although the persistence layer can also be scaled horizontally it's not so easy has scaling the services, it requires changes in the service layer in the way as communication is established, this needs to be improved in the future has the platform grows;

i) The simplicity of the platform can be subjective, but based on the requirements and final solution, this work concludes that this objective has been accomplished, the platform can be maintained by a small number of people without deep technical knowledge of the architecture and technology, changing requirements or adding new services though, needs to be done by a developer with technical knowledge of both. Still, for this kind of solution, the platform has very few dependencies and a simple architecture making it easier for bringing new people into the project.

4.2. Limitations and Future Work

Following is a general view of the most prominent improvements that can or should be made to this work.

4.2.1. HL7 Support

The inclusion of these standards would also add complexity to the platform and this, added to the fact that very few services currently needed it, is why it was not considered essential to complete the platform. It could, however, be proven useful for integration with some organizations in the future, so it's important to consider.

4.2.2. HTTP/2

Like already said in chapter 2.3.2.1 it's very important, mainly because of performance issues, to keep an eye on the adoption of HTTP/2 because this can increase significantly the performance of the overall system, since HTTP and RESTful communications are core functionalities of this platform. Play Framework already supports HTTP/2 but it’s not yet ready for a production environment.

There can be two approaches when adopting HTTP/2, one is to migrate the existing services from HTTP/1.1 to HTTP/2 and the other is to support both protocols for a transitional period of time, from a safe bet perspective the second approach is the better one, but this is subject to additional testing has the adoptions of HTTP/2 evolves in general and in the Play Framework and Akka HTTP implementation.

4.2.3. Apache Camel

Apache Camel[79] is an open-source framework for integrating known Enterprise Integration Patterns[80], based on the book Enterprise Integration Patterns: Design, Building and Deploying Messaging Solution by Gregor Hohpe and Bobby Woolf. Its engine is a rule-based routing and mediation and it’s message oriented, connecting different sources to different destination while also transforming message formats. It focuses on making integration easier and provides DSLs to facilitate it’s use (Camel, 2017).

As an advantage, this could greatly improve the flexibility of the platform, since there is no way to know in advance all the services and message formats that it would need to support. It supports, by default, Java and Scala which is convenient for this platform.

As a disadvantage, for someone that never worked with Camel thou, it requires a steep learning curve until a satisfactory production ready solution is developed and it also adds complexity to the platform, also Akka already provides some of the features that Camel provides, like routing and mediation patterns, so at least for the start of the platform, it makes more sense to take advantage of the Akka framework that is already a core component of the platform then to add an additional external one.

4.2.4. JSON Coast-to-Coast Design

This work is mainly developed with Scala with functional programming in its core, but there also a lot of object oriented programming, for example the models are designed from an object oriented perspective, so a typical request that sends data to the database can pass through the following steps:

1. Get the request from the client and extracts the JSON message;
2. Converts the JSON to the objects model;
3. Do some work on the model objects;
4. Transform the objects model to the SQL insert/update scripts;

---

79 http://camel.apache.org/
80 http://www.enterpriseintegrationpatterns.com/patterns/messaging/toc.html

5. Send to database.

From these steps, it possible to see that the convention from JSON to OO could be avoided in most cases, by using databases with document structured data, like MongoDB. The JSON messages could be sent directly to the database without first transforming them to model objects, so the steps would translate into (Lightbend, 2017):

1. Get the request from the client and extract the JSON message;
2. Transform JSON if needed;
3. Send JSON to Database.

This way, the flow can be treated in a fully asynchronous and non-blocking way and the overhead of maintaining the model classes and transforming JSON into object and vice-versa is eliminated (Lightbend, 2017).

Although this presents a clear advantage over the traditional model, it also means that a different design approach must be considered from the beginning, and since part of the platform consisted in integrating existing models, this approach was not a priority, but future work should consider this approach for new services or even for the refactoring of existing ones.

4.2.5. Security Concerns

Security of the platform should be a continuous effort, has new services are supported and new features are added, also the security checks and mechanisms should evolve. At its current state, well known security threats were address and mechanisms where put in place to secure the platform, but this is not enough and more testing needs to be done to give the shareholders more assurances that their data is protected.

Future work here can involve the use of external tools that help in detecting security vulnerabilities and/or acquire the help of an external entity specialized in such matters.

End-to-End-Encryption needs to be improved significantly, its implementation needs to be more seamless for service consumers. This work is already in progress and the solution is based in the Diffie-Hellman Key Exchange, which is a mathematical algorithm that allows two endpoints to generate an identical shared secret and uses private/public key crypto to achieve this (Palmgren, 2006).

Regulation demanded by CNPD (see chapter 2.6), although not mandatory at this time, also requires significant change, so it’s very important to start addressing its requirements

4.2.6. Abuse Protection

In the current state of this work the platform already provides some basic, but none the less essential, abuse protection, but there are yet several improvements to be made here:

- User based request rate limiter: The currently rate limiter implementation only distinct sources based on its IP address. User based request limiter will give more flexibility and, by consequence, better protection against abuse;
- Concurrent requests limiter;
- Worker utilization load shedder: A load shedder differs slightly from a rate limiter in that it makes its decisions based on the whole system rather than just on a component or access source, it's useful in a situation of emergency where critical/core systems must be kept operational while the rest of the system might be offline;
- Evolve to a distributed memory based storage, like Redis.

4.2.7. Akka Persistence

As the system grows, scaling the data side of the platform becomes even more important, Akka Persistence[81] offers a resilient, scalable actors based approach framework to implement persistence solutions. Akka Persistence already has methods in place to help with Event-Source with CQRS implementation.

4.2.8. Akka Typed

Akka Typed[82] can be viewed has the successor for regular Akka actors, and it basically means the development of static typed Akka actors. It makes sense to evolve the developing model of the Akka actors to Akka typed, although currently, Akka typed in still in its early stages and Lightbend advises that it might be subject to change, for this reason it's not the perfect time to adopt such technology, but it's something to keep in mind for the future.

4.2.9. Turn Router Into a Play Framework Module Or Library

There are a lot of popular modules that extend Play's Framework functionality that exist today, but currently there exists none that can transform Play into a Reverse Proxy and API Gateway. Of course, there exists other solutions, and probably better ones, for bigger designs, like using NGINX as the Reverse Proxy, but this is always a more complex solution and probably more time consuming because of the learning curve necessary to implement and manage NGINX. Having a Reverse Proxy and API Gateways as a Play module will allow to rapidly implement this functionality in system that already uses the Play Framework. This would also be an opportunity to contribute back to the community and hopefully have its contribute in return.

This would involve some changes into the actual component, to make it more general purpose, and time, to dedicate to the project has it will certainly be needed to help anyone who wants to use the module and to evolve the project, fix bugs, etc.

81 https://doc.akka.io/docs/akka/current/scala/persistence.html
82 https://doc.akka.io/docs/akka/current/scala/typed-actors.html

4.2.10. Performance Testing

Future performance testing can be automated by integrating the Vegeta library into the platform, this will allow to incorporate the performance tests and reporting into the pipeline.

4.3. Final Thoughts

Although there exists platforms that could fulfill the needs of this work, developing the MHP was the best fit. Financially its development was very affordable, because all technology used is free for commercial use and it was all implemented by a single developer, maintaining it is easy because of its simplicity and high-availability and all this wouldn't be true, if not for a couple of key frameworks, mainly the Play Framework and the Akka Framework, which provided an abstraction layer between the developer and the more complex concerns and, at the same time, were very easy to configure and set up and, worked has expected.

5. References

(Alexander, 2003)	Stakeholders - Who Is Your System For, March 2003, Ian Alexander, http://www.scenarioplus.org.uk/papers/stakeholders/stakeholders.html.
(Allee, 2012)	Value Network Analysis," 2012, Allee, Verna, http://www.uio.no/studier/emner/matnat/ifi/INF5120/v12/undervisningsmateriale/ValueNetworks2012023.pdf.
(Amazon, 2017)	AWS | Amazon ElastiCache – in-Memory Data Store and Cache, 2017, Amazon AWS, https://aws.amazon.com/elasticache/.
(Belshe et al., 2015)	RFC 7540 - Hypertext Transfer Protocol Version 2 (HTTP/2), May 2015, Belshe, M., BitGo, R. Peon, Google, Ed. M. Thomson, and Mozilla, https://tools.ietf.org/html/rfc7540.
(Bonér et al., 2014)	The Reactive Manifesto V2.0, September 16, 2014, Jonas Bonér, Dave Farley, Roland Kuhn, and Martin Thompson, Accessed February 2017, http://www.reactivemanifesto.org.
(Bonér, 2017)	*Reactive Microsystems - The Evolution of Microservices a Scale*, 2017, Jonas Bonér.
(Brikman, 2013)	Play Framework: Async I/O without the Thread Pool and Callback Hell, March 27, 2013, Yevgeniy Brikman, https://engineering.linkedin.com/play/play-framework-async-io-without-thread-pool-and-callback-hell.
(Churchville, 2017)	The Reality of Microservices Adoption and the Limits of Your Monolith, July 2017, Fred Churchville, http://searchmicroservices.techtarget.com/feature/The-reality-of-microservices-adoption-and-the-limits-of-your-monolith?utm_medium=EM&asrc=EM_NLN_80572968&utm_campaign=20170728_The%20reality%20of%20microservices%20adoption&utm_source=NLN&track=NL-1806&ad=915729&src=915729.
(Coleman, 2016)	Containers are not VMs, 2016, Mike Coleman, Accessed July 23, 2017, https://blog.docker.com/2016/03/containers-are-not-vms/.
(CNPD, 2017)	10 Medidas Para Preparar RGPD CNPD, January 28, 2017, Comissão Nacional de protecção de Dados, https://www.cnpd.pt/bin/rgpd/10_Medidas_para_preparar_RGPD_CNPD.

	pdf.
(Flannelly et al., 2014)	Fundamentals of Measurement in Health Care Research.: Sistema de Descoberta Para FCCN, 2014, FLANNELLY, LAURA T., KEVIN J. FLANNELLY, and KATHERINE R. B. JANKOWSKI.
(Fowler, 2005)	Event Sourcing," 2005, Martin Fowler, https://martinfowler.com/eaaDev/EventSourcing.html.
(Fowler, 2011)	CQRS, July 14, 2011, Martin Fowler, https://martinfowler.com/bliki/CQRS.html.
(Fowler, 2014)	BoundedContext," 2014, Martin Fowler, https://martinfowler.com/bliki/BoundedContext.html.
(Gartner, 2017)	Application Platforms Reviews, 2017, Gartner, https://www.gartner.com/reviews/market/application-platforms-reviews/
(Gudgin et al., 2005)	XML-Binary Optimized Packaging, January 25, 2005, Gudgin, Martin, Noah Noah, Mark Nottingham, and Hervé Ruellan, https://www.w3.org/TR/2005/REC-xop10-20050125/#introduction.
(Gupta, 2014)	REST vs WebSocket Comparison and Benchmarks, February 24, 2014, Arun Gupta, http://blog.arungupta.me/rest-vs-websocket-comparison-benchmarks/.
(Halter and Shepherd, 2012)	Actor-Based Concurrency: Implementation and Comparative Analysis With Shared Data and Locks, 2012, Halter, Alex, and Randy Shepherd, http://cs.nyu.edu/~lerner/spring12/Preso01-Actors.pdf.
(Hameri and Nitter, 2000)	Engineering data management through different breakdown structures in a large-scale project, July 7, 2000, Hameri, A.P., and P. Nitter.
(Hayes et al., 2011)	Improving HTTP Performance Using 'Stateless' TCP, June 3, 2011, David A Hayes, Michael Welzl, Grenville Armitage, and Mattia Rossi.
(H. Behringer, 2009)	End-to-End Security - The Internet Protocol Journal, Volume 12, No.3 – Cisco, September 2009, H. Behringer, Michael. http://www.cisco.com/c/en/us/about/press/internet-protocol-journal/back-issues/table-contents-45/123-security.html.
(HL7, 2017)	Health Level Seven International - Homepage, 2017, HL7, https://www.hl7.org/.
(IBM, 2017a)	IBM Passport Advantage Express, 2017, IBM. https://www-112.ibm.com/software/howtobuy/buyingtools/paexpress/Express?P0=E1&part_number=D58AKLL,D58AELL,D06UVLL,D04Q1LL,D04Q3LL,D0BA1LL,D0B3HLL,D0B5HLL,D0B2XLL,D0GALLL,D0GB0LL&catalogLocale=en_US&locale=en_US&country=USA&PT=html.
(IBM, 2017b)	IBM Passport Advantage Express. Accessed February 23, 2017, IBM, https://www-112.ibm.com/software/howtobuy/buyingtools/paexpress/Express?P0=E1&part_number=D56P3LL,D56MWLL,D03S0LL,D03S4LL,D03S2LL,D06UPLL,D0BLZLL,D0GH4LL,D0GH1LL,D0WE0LL,D0WG2LL,D0WG4LL,D0WG6LL&catalogLocale=en_US&locale=en_US&country=USA&PT=html.
(IBM, 2017c)	IBM Knowledge Center - Service Component Architecture (SCA), 2017, IBM, https://www.ibm.com/support/knowledgecenter/en/SSGMCP_5.3.0/com.ibm.cics.ts.applicationprogramming.doc/bundleinterface/sca.html.

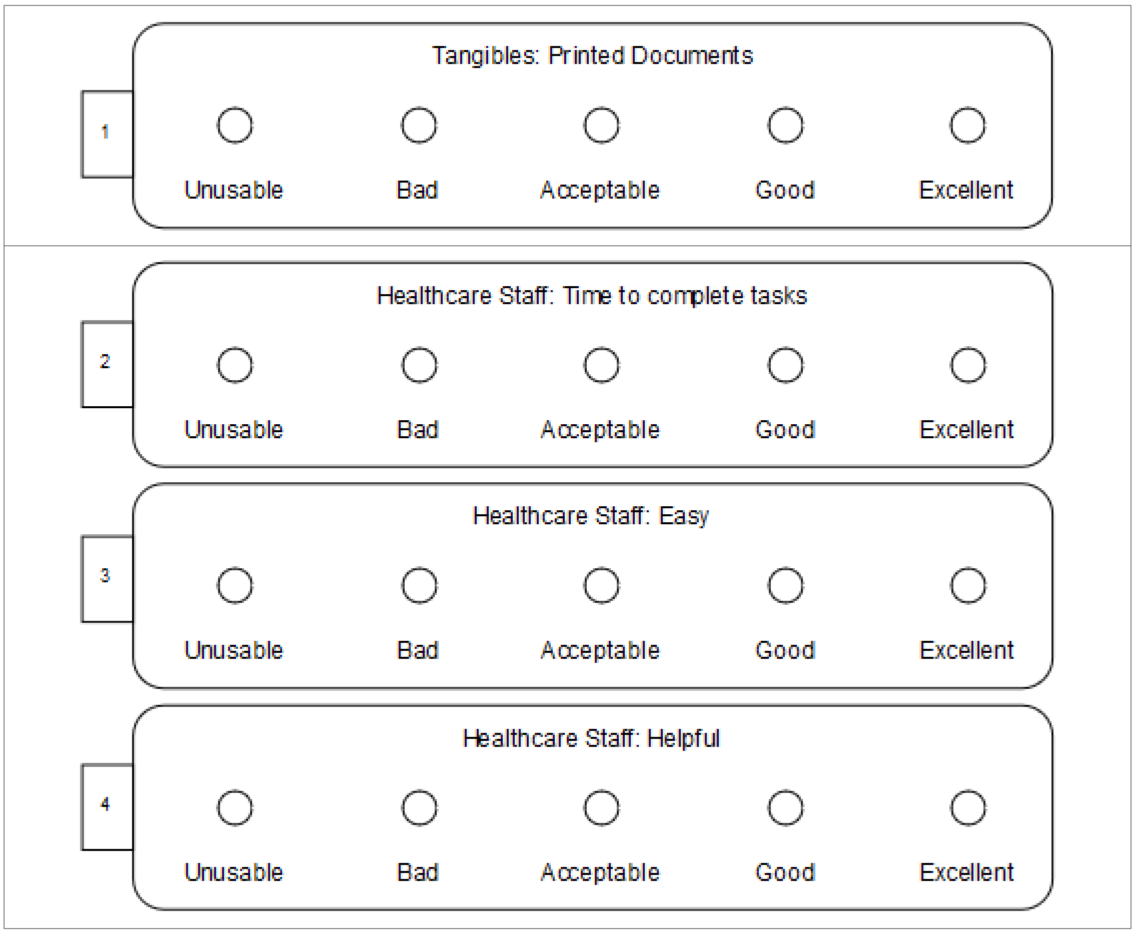
(IBM, 2017d)	Integration Bus on Cloud - IBM Integration, 2017, IBM, https://developer.ibm.com/integration/docs/integration-bus-on-cloud/.
(Jackson, 2016)	Create and Execute Business Rules in IBM Integration Bus, November 22, 2016, Callum Jackson, https://www.ibm.com/developerworks/bpm/bpmjournal/1308_jackson/1308_jackson.html.
(Jakupovic et al., 2010)	Application of Analytic Hierarchy Process (AHP) to Measure the Complexity of the Business Sector and Business Software, May 19, 2010, Jakupovic, Alen, Mile Pavlic, and Sanja Candrlic.
(Kang et al., 2016)	The Design and Analysis of a Secure Personal Healthcare System Based on Certificates, November 14, 2016, Kang, Jungho, Hague Chung, Jeongkyu Lee, and Jong Hyuk Park.
(Li et al., 2012)	A Comparative Study between Soft System Bus and Enterprise Service Bus, 2012, LI, Gang, Xiao Jian, Chun Li, Sen Li, and Jingde Cheng.
(Lightbend, 2017a)	Lightbend Telemetry v. 2.5.0, 2017, Lightbend, http://developer.lightbend.com/docs/cinnamon/latest/home.html.
(Lightbend, 2017b)	Lightbend Tech Hub / Documentation, 2017, Lightbend, http://developer.lightbend.com/docs/.
(Lightbend, 2017c)	Supervision and Monitoring - Akka Documentation, 2017, Lightbend, http://doc.akka.io/docs/akka/snapshot/scala/general/supervision.html.
(Lightbend, 2017d)	*ScalaJsonTransformers - 2.6.X*, 2017, Lightbend, https://www.playframework.com/documentation/2.6.x/ScalaJsonTransformers.
(Lightbend, 2017e)	Cross-Site Request Forgery (CSRF) - OWASP, 2017, Lightbend, https://www.owasp.org/index.php/Cross-Site_Request_Forgery_%28CSRF%29.
(Lukác et al., 2016)	A Process-Oriented Service Infrastructure for Networked Enterprises, November 20, 2016, Lukác, Gabriel, Tomáš Sabol, Martin Tomášek, and Karol Furdík.
(Lupo, 2015)	A Fuzzy Framework to Evaluate Service Quality in the Healthcare Industry: An Empirical Case of Public Hospital Service Evaluation in Sicily" December 18, 2015, Toni Lupo.
(Man et al., 2012)	Research on Bulk Data Transfer on OSS Enterprise Service Bus, June 19, 2012, MAN Yi, Wen ZHANG, Chen-hui DU, and Yang-fa PAN.
(Mincer-Daszkiewicz, 2015)	Responsive, Resilient, Elastic and Message Driven System Solving Calability Problems of Course Registrations, Accessed February 23, 2017, Mincer-Daszkiewicz, Janina, http://dspacecris.eurocris.org/bitstream/11366/464/1/EUNIS2015_submission_29.pdf.
(Ming-zhe, 2013)	Design on Enterprise Service Bus Message Conversion Protocol Based on XSLT, August 20, 2013, Ming-zhe, YU
(Mulesoft, 2017)	Download Mule ESB Enterprise | MuleSoft, 2017, MuleSoft, https://www.mulesoft.com/platform/soa/mule-esb-enterprise.
(OWASP, 2017)	Cross-Site Request Forgery (CSRF) - OWASP," June 20, 2017, OWASP, https://www.owasp.org/index.php/Cross-Site_Request_Forgery_%28CSRF%29.

(Palmgren, 2006)	Diffie-Hellman Key Exchange - A Non-Mathematician's Explanation, October 2006, Keith Palmgren.
(Raghavan et al., 2007)	Timeseries.Eps - P337-Raghavan, August 2007, Barath Raghavan, Kashi Vishwanath, Sriram Ramabhadran, Kenneth Yocum, and Alex C. Snoeren, http://delivery.acm.org/10.1145/1290000/1282419/p337-raghavan.pdf?ip=193.136.62.5&id=1282419&acc=PUBLIC&key=2E5699D25B4FE09E%2E6A9D8E1D9CC021F6%2E4D4702B0C3E38B35%2E4D4702B0C3E38B35&CFID=810325093&CFTOKEN=42832574&__acm__=1505664732_453b86b4e760d943a3466b1a2fc066a7#URLTOKEN#.
(Rao, 2011)	Denial of Service Attacks and Mitigation Techniques: Real Time Implementation with Detailed Analysis, 2011, Subramani Rao, https://www.sans.org/readingroom/whitepapers/detection/denial-service-attacks-mitigation-techniques-real-time-implementation-detailed-analysi-33764.
(RedHat, 2014)	Fuse Service Works Compared Mule Esb, April 2014, RedHat, https://www.redhat.com/en/files/resources/en-rhjb-fuse-service-works-compared-mule-esb-11851667.pdf.
(RedHat, 2015)	Fuse Compared with Ibm Websphere Enterprise Service Bus," 2015, RedHat, https://www.redhat.com/en/files/resources/en-rhjb-fuse-compared-with-ibm-websphere-enterprise-service-bus-10570127.pdf.
(Rivetti, 2016)	Load-Aware Shedding in Stream Processing Systems," 2016, Nicoló Rivetti, Yann Busnel, and Leonardo Querzoni, http://delivery.acm.org/10.1145/2940000/2933311/p61-rivetti.pdf?ip=193.136.62.5&id=2933311&acc=ACTIVE%20SERVICE&key=2E5699D25B4FE09E%2E6A9D8E1D9CC021F6%2E4D4702B0C3E38B35%2E4D4702B0C3E38B35&CFID=985788164&CFTOKEN=70330353&__acm__=1505662111_ab28c992f1aaa0a964393d70403d29f1.
(Shahriar et al., 2010)	Client-Side Detection of Cross-Site Request Forgery Attacks, 2010, Hossain Shahriar and Mohammad Zulkernine.
(Simec, 2014)	Comparison of JSON and XML Data Formats," September 19, 2014, Alen Simec and Magdalena Maglicic.
(Stafford and McKenzie, 2014)	Moving Away from ESBs to Microservices, 2014, Stafford, Jan, and Cameron McKenzie, http://searchmicroservices.techtarget.com/video/Moving-away-from-ESBs-to-microservices.
(Sudhodanan, 2017)	Large-Scale Analysis & Detection of Authentication Cross-Site Request Forgeries, 2017, Avinash Sudhodanan, Roberto Carbone, Luca Compagna, Nicolas Dolgin, Alessandro Armando, and Umberto Morelli.
(WSO2, 2017a)	WSO2 Enterprise Integrator Documentation - Enterprise Integrator 6.1.1 - WSO2 Documentation, 2017, WSO2, https://docs.wso2.com/display/EI611.
(WSO2, 2017b)	Integration - Cloud, 2017, WSO2, http://wso2.com/integration/cloud/#pricing.
(WSO2, 2017c)	Price List, August 1, 2017, WSO2.
(Zhu et al., 2015)	Microarchitectural Implications of Event-Driven Server-Side Web Applications, 2015, Zhu, Yuhao, Daniel Richins, Halpern Matthew, and Vijay Janapa Reddi, http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7856643.

6. Annexes

6.1. Perceived Value Ordinal Scale

The following ordinal scales were sent to the stakeholders in order to calculate the perceived value of this platform initial state. Only the external stakeholders were included In this study, has the internal ones are biassed.



1 Tangibles: Printed Documents

○ Unusable ○ Bad ○ Acceptable ○ Good ○ Excellent

2 Healthcare Staff: Time to complete tasks

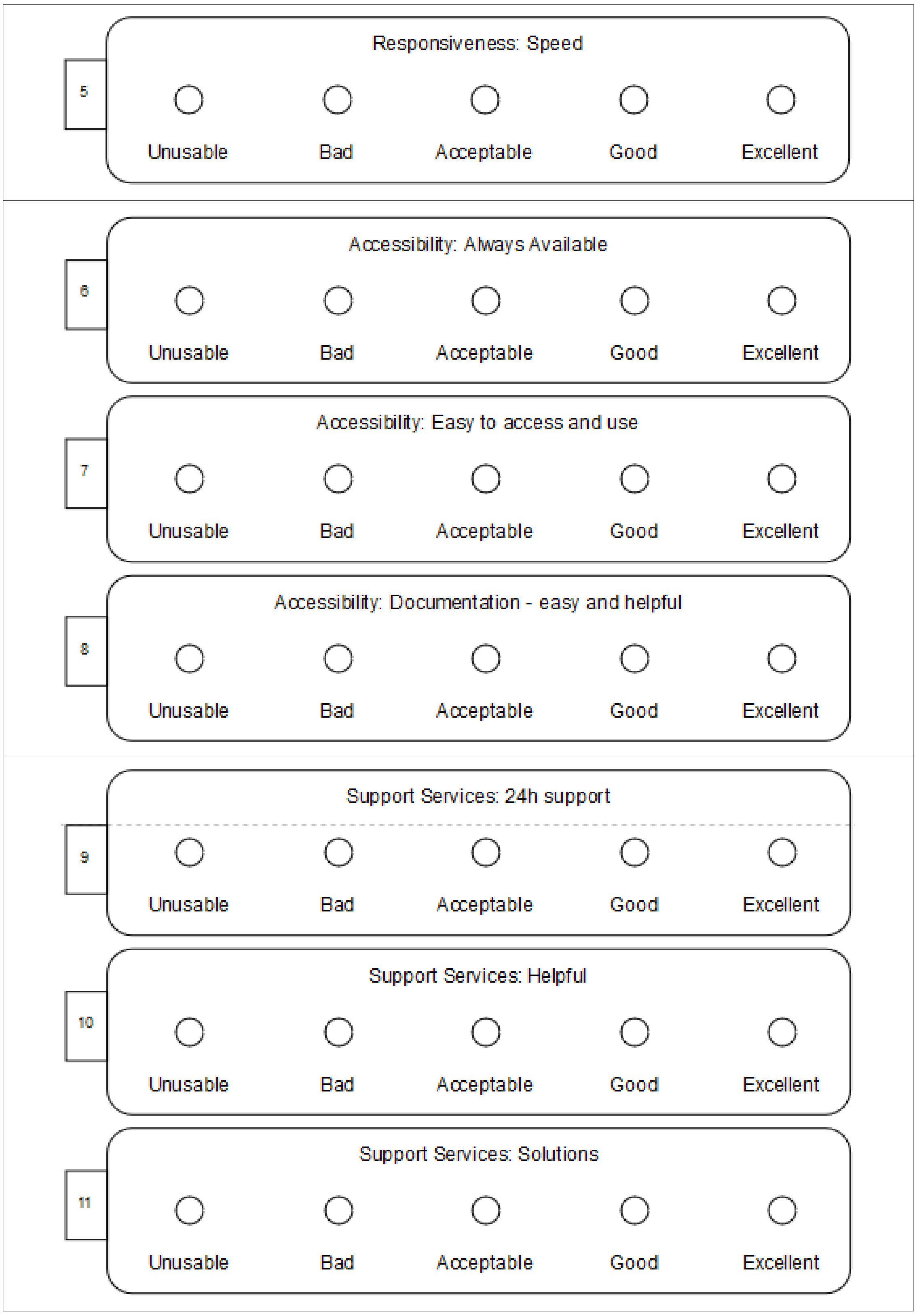
○ Unusable ○ Bad ○ Acceptable ○ Good ○ Excellent

3 Healthcare Staff: Easy

○ Unusable ○ Bad ○ Acceptable ○ Good ○ Excellent

4 Healthcare Staff: Helpful

○ Unusable ○ Bad ○ Acceptable ○ Good ○ Excellent



5 Responsiveness: Speed

○	○	○	○	○
Unusable	Bad	Acceptable	Good	Excellent

6 Accessibility: Always Available

○	○	○	○	○
Unusable	Bad	Acceptable	Good	Excellent

7 Accessibility: Easy to access and use

○	○	○	○	○
Unusable	Bad	Acceptable	Good	Excellent

8 Accessibility: Documentation - easy and helpful

○	○	○	○	○
Unusable	Bad	Acceptable	Good	Excellent

9 Support Services: 24h support

○	○	○	○	○
Unusable	Bad	Acceptable	Good	Excellent

10 Support Services: Helpful

○	○	○	○	○
Unusable	Bad	Acceptable	Good	Excellent

11 Support Services: Solutions

○	○	○	○	○
Unusable	Bad	Acceptable	Good	Excellent

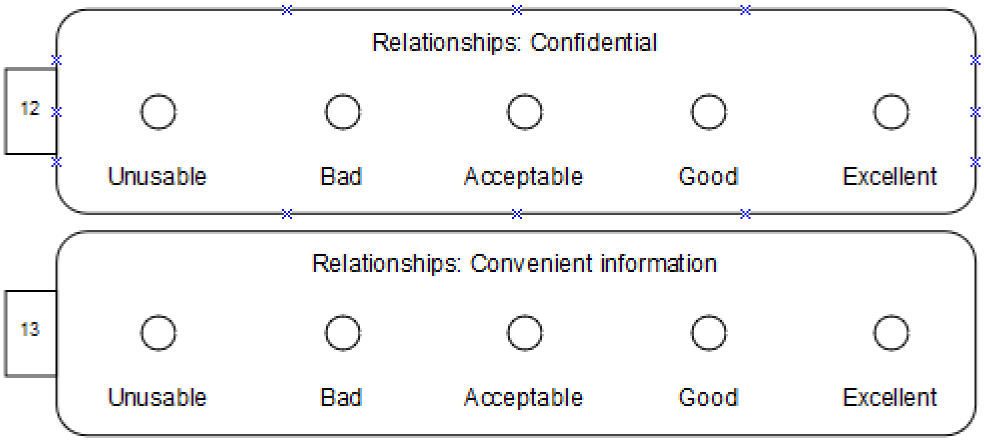


Figure 36: Ordinal scales sent to the stakeholders

According to the category different stakeholder were chosen, the only not all of them responded, so the results shown here correspond to only the ones that responded. The identity of the stakeholders are protected for privacy reasons, since this project is of a commercial nature. The number of responses/scales corresponds to the total of scales that had a response. The medium value p/category represents the median of each category, which gives a closer appreciation of each category. The median value is calculated based on the sum of the value of the responses divided by the number of responses.

Table 49: Stakeholders response to the perception study

Stakeholder	Scales	Nº of Responses/Scales	Median Value p/Category	Median Value
Investors	1, 6-8, 9-11	14	1:4, 6-8:4, 9-11:5	4.3
Laboratories	5, 6-8, 9-11, 12-13	26	5:5, 6-8:4, 9-11:4, 12-13:3	4
Medication Suppliers	1, 5, 6-8, 9-11, 12-13	30	1:2, 5:4, 6-8:4, 9-11:3, 12-13:2	3
Dictionary Suppliers	9-11	3	9-11:4	4
Financial Institutions	1, 6-8, 9-11	14	1:3, 6-8:5, 9-11:4	4
Patients	5, 12-13	48	5:4, 12-13:4	4
Pharmacy Associations	1, 5, 6-8, 9-11	8	1;4, 5:5, 6-8:5, 9-11:4	4.5
Pharmacies	1, 2-4, 5, 6-	52	1:2, 2-4:4, 5:4, 6-8:5, 9-	3.7

	8, 9-11, 12-13		11:3, 12-13: 4	
Regulatory Entities	9-11, 12-13	-	-	-
				Median: 3.94

Based on the results above, there are some important consideration:

- The lower value of the Medication Suppliers if mostly due to the fact that at this stage, reporting is very poor and because the encryption method is not transparent it requires that the supplier requires more technical knowledge which in some cases might result in outsourcing which is not ideal in this cases.
- The lower value of the Pharmacies is due too the fact that they depend a lot of reporting, and reporting was not a major priority of this platform initial stage, also the support staff was not all up to date with the new platform, so, some support questions might have taken a little longer to respond, because they needed to escalate inside the company, and pharmacies are a stakeholder which is used to a higher level of attention.
- Only one stakeholder from Pharmacy Associations and Dictionary Suppliers responded, so this justifies the lower number of responses. At this stage, Pharmacy Associations represent two entities while the Dictionary Suppliers only represent one.
- At this stage there was no direct usage of the platform from Regulatory Entities stakeholder.
- The final result is not the most optimum result, although at this stage and for several non technical reasons, like time to deliver, this was expected, the most critical areas where identified, and this is where the next efforts should focus to improve the stakeholder perceived value.

6.2. Rate Limiting Configuration Example

The following table shows a configuration example of the rate limiting implementation.

Table 50: Rate limiting configuration example

Order	Description

PlayGuard Specific	`playguard {`
	`  clientipheader = "X-Forwarded-For"`
	`  # required for the global GuardFilter`
	`  filter {`
	`    enabled = true`
	`    global {`
	`      bucket {`
	`        size = 100`
	`        rate = 100`
	`      }`
	`    }`
	`    ip {`
	`      whitelist = ["127.0.0.1"]`
	`      blacklist = ["3.3.3.3", "4.4.4.4"]`
	`      bucket {`
	`        size = 50`
	`        rate = 50`
	`      }`
	`    }`
	`  }`
	`}`

Platform Specific	`app {`
	`  playguard {`
	`    ip {`
	`      services {`
	`        default {`
	`          # nr of requests allowed`
	`          nrOfRequests = 20`
	`          # rate at which tokens are added to the bucket in seconds`
	`          tokenRate = 3`
	`          logPrefix = "default-ip-rate-limit"`
	`        }`
	`        auth {`
	`          # nr of requests allowed`
	`          nrOfRequests = 30`
	`          # rate at which tokens are added to the bucket in seconds`
	`          tokenRate = 2`
	`          logPrefix = "auth-ip-rate-limit"`
	`        }`
	`      }`
	`    }`
	
	`    httpError {`
	`      services {`
	`        default {`
	`          nrOfRequests = 20`
	`          tokenRate = 3`
	`          logPrefix = "default-http-rate-limit"`
	`        }`

```
            }
         }
      }
   }
```

6.3. API Gateway Code

The following code is the API Gateway implementation.

```
def proxy(service: String, resource: String, id: Option[String]) =
  (Action
    andThen rateLimitServ.getKeyRateFilter(service)).async { implicit req =>
    andThen rateLimitServ.getHttpErrorRateFilter(service)).async { implicit req
=>

    try {
      // 1. Get service virtual server if exists
      val virtualServer = proxyServ.getVirtualServer(service)
      val virtualServerAuth = if (service.equals(this.authService)) virtualServer
else proxyServ.getVirtualServer(this.authService)

      // 2. Finding out if it uses HTTPS
      val https = proxyServ.usesHTTPS(service)
      val httpsAuth = if (service.equals(this.authService)) https else
proxyServ.usesHTTPS(this.authService)

      for {
      // 2. Validate authentication/authorization
        authenticated <- authServ.validate(service, virtualServerAuth, httpsAuth,
"isAuthenticated", req)

        // 3. Ask router for backend address (router uses load-balancing)
        backend <- proxyServ.getBackend(
          authenticated,
          service,
          if (id.isDefined) true else false,
          if (id.isDefined) id.get else null)

        // 4. Proxy requests to backend
        wsResponse <- backend match {
          case null =>
            val emptyResponse: WSResponse = null
            Future(emptyResponse)

          case _ =>
            proxyServ.relayRequest(backend, virtualServer, https, service,
resource, req)
        }

      // 5. Process Response from backend and Reply to original request
      } yield proxyServ.processResponse(service, authenticated, wsResponse)

    } catch {
      case e: UnknownHostException =>
        log.error(s"Couldn't determine auth url endpoint: $e")

Future.successful(ServiceUnavailable(StatusResult(MsgStatus.SERVICE_UNAVAILABLE).
toJson))
      case e: Exception =>
        log.error("System error", e)

Future.successful(InternalServerError(StatusResult(MsgStatus.INTERNAL_SERVER_ERRO
R).toJson))
```

```
    }
  }
```

Code 16 – API Gateway Code

The following code is the API Gateway implementation.

```
  def relayRequest(backend: String, virtualServer: String, https: Boolean, service: 
  String, resource: String,
                     req: Request[AnyContent], includeQueryString: Boolean = true, 
  httpMethod: String = null
                    ): Future[WSResponse] = {

    val url =
      if (resource == null || resource.isEmpty)
        s"${if (https) "https" else "http"}://${backend}/api/${service}"
      else
        s"${if (https) "https" else "http"}://${backend}/api/${service}/$
  {resource}"

    log.debug(s"Relaying ${req.method} to $url")

    var wsr: WSRequest = ws.url(url)

    // Add origin request headers
    wsr = wsr.withHttpHeaders(
      req.headers.toSimpleMap.filterNot(x =>
        x._1.equals("Content-Type") || x._1.equals("Content-Length")
      ).toList: _*)

    // add virtual server for https validation
    if (virtualServer != null) {
      log.debug(s"adding virtual server $virtualServer")
      wsr = wsr.withVirtualHost(virtualServer)
    }

    // Add query strings from original request
    if (includeQueryString) {
      req.queryString.foreach { qs =>
        wsr = wsr.withQueryStringParameters((qs._1, qs._2.mkString(",")))
      }
    }

    // Add body
    if (req.hasBody) {
      req.headers.get("Content-Type") match {

        case Some("application/x-www-form-urlencoded") =>
          wsr = wsr.withBody(req.body.asFormUrlEncoded.get)

        case Some("application/json") =>
          wsr = wsr.withBody(req.body.asJson.get)

        case _ =>
          log.warn(s"application content not supported: ${req.headers.get("Content-
  Type").getOrElse("NoneFound")}")
          wsr = wsr.withBody(req.body.asText.get)
      }
    }
    wsr = wsr
      // Add method
      .withMethod(if (httpMethod == null) req.method else httpMethod) // If
  httpMethod is null then use same http method as from request

    // Send request
    log.info(s"Sending POST: ${wsr.uri} with Content-Type: $
  {req.headers.get("Content-Type")}")
    wsr.execute
  }
```

Code 17 – API Gateway Proxy Code

6.4. Reverse Proxy Code

The following code corresponds to the Reverse Proxy process.

```
def proxy(service: String, resource: String) = Action.async { implicit req =>

  try {
    val site = req.headers.get("Host").getOrElse("")
    log.debug(s"Request for site $site")

    // Identify backend from Host
    val backend = sitesServices.getAddress(site, defaultRoute)
    log.debug(s"Backend - $backend")

    for {
      // Proxy requests to backend
      result <- backend match {
        case "" =>
          Future(null: Result)

        case this.defaultRoute => // this condition is very important because it
avoids a cyclic redundancy
          relayRequest(backend, null, null)

        case _ =>
          relayRequest(backend, service, resource)
      }

    // Process Response from backend AND Reply to original request
    } yield (result) match {

      // if didn't received response from backend
      case response if response == null =>
        log.error(s"Service unavailable, didn't receive response from backend
$service")
        ServiceUnavailable(StatusResult(MsgStatus.SERVICE_UNAVAILABLE).toJson)

      // if received a Ok message from backend service
      case response if response.header.status == 200 =>
        response

      // if received a NOT Ok message from backend service
      case response if response.header.status != 200 =>
        response

      case x: Any =>
        log.error(s"Unsupported reply: ${x}")
        NotFound(StatusResult(MsgStatus.NOT_FOUND).toJson)
    }
  }

  catch {
    case e: UnknownHostException =>
      log.error(s"Couldn't determine auth url endpoint: ${e}")

Future.successful(ServiceUnavailable(StatusResult(MsgStatus.SERVICE_UNAVAILABLE).
toJson))
```

```
    case e: Exception =>
      log.error("System error", e)

Future.successful(InternalServerError(StatusResult(MsgStatus.INTERNAL_SERVER_ERRO
R).toJson))
  }
}


def relayRequest(backend: String, service: String, resource: String,
                 includeCustomHeaders: Boolean = true, includeQueryString:
Boolean = true,
                 httpMethod: String = null
                )(implicit req: Request[AnyContent]): Future[Result] = {

  // Construct URL
  val url = s"${backend}" +
    (if (service == null || service.isEmpty) "" else s"/${service}") +
    (if (resource == null || resource.isEmpty) "" else s"/${resource}")

  log.debug(s"Relaying ${req.method} to $url")

  // Create the request to the upstream server
  var proxyRequest =
    ws
      // Add URL
      .url(url)

      // Add original request method
      .withMethod(req.method)

      // Add virtual Host
      .withVirtualHost(conf.getOptional[String]
(s"app.services.auth.virtualServer").getOrElse(null))

      // Add HTTP Headers
      .withHttpHeaders(req.headers.toSimpleMap.toList: _*)

      // add revere-se proxy header to tell the service who is requesting the
service
      .addHttpHeaders("X-Forwarded-For" -> req.remoteAddress)

      // Add Query String parameters
      .withQueryStringParameters(req.queryString.toSeq.map(qs => (qs._1,
qs._2.mkString(","))): _*)

      // Add Cookies
      .withCookies(
        req.cookies.toSeq.map(x =>
          new DefaultWSCookie(x.name, x.value, x.domain, Option(x.path),
            if (!x.maxAge.isDefined) None else Some(x.maxAge.get.toLong),
            x.secure, x.httpOnly)): _ *)

  // Add body
  if (req.hasBody) {
    req.headers.get("Content-Type") match {

      case Some("application/x-www-form-urlencoded") =>
        proxyRequest = proxyRequest.withBody(req.body.asFormUrlEncoded.get)

      case Some("application/json") =>
        proxyRequest = proxyRequest.withBody(req.body.asJson.get)
      case _ =>
        log.warn("application content not supported")
        proxyRequest = proxyRequest.withBody(req.body.asText.get)
    }
  }


  // Send Request and Stream Results
  proxyRequest.stream().map { resp =>
    log.warn("processing response")

    // Check that the response was successful.
    if (resp.status == 200) {

      // Get the content type.
      val contentType = resp.headers.get("Content-Type")
        .flatMap(_.headOption)
        .getOrElse("application/octet-stream")

      // Remove "Transfer-Encoding" header if present
      val headers = resp.headers.filterKeys(_ != "Transfer-Encoding")

      // If there's a content length, send that, otherwise return the body
chunked.
      resp.headers.get("Content-Length") match {
        case Some(Seq(length)) =>
          log.debug("Streamed response")

          Ok.sendEntity(HttpEntity.Streamed(resp.bodyAsSource,
Some(length.toLong), Some(contentType)))

            .withHeaders(headers.mapValues(_.mkString(",")).toSeq.filterNot(x =>
x._1.equals("Content-Type") || x._1.equals("Content-Length")): _*)

            .withCookies(resp.cookies.map(x =>
              Cookie(x.name, x.value,
                if (!x.maxAge.isDefined) None else Some(x.maxAge.get.toInt),
                x.path.getOrElse("/"), x.domain, x.secure, x.httpOnly)): _*)

        case _ =>
          log.debug("Chunked response")

          Ok.chunked(resp.bodyAsSource)

            .withHeaders(headers.mapValues(_.mkString(",")).toSeq.filterNot(x =>
x._1.equals("Content-Type") || x._1.equals("Content-Length")): _*)

            .withCookies(resp.cookies.map(x =>
              Cookie(x.name, x.value,
                if (!x.maxAge.isDefined) None else Some(x.maxAge.get.toInt),
                x.path.getOrElse("/"), x.domain, x.secure, x.httpOnly)): _*)

            .as(contentType)
      }
    } else {
      BadGateway
    }
  }
}
```

Code 18 – Reverse Proxy Code

6.5. Play and Akka Main Configuration Files Example

The following example shows a standard Play Framework configuration file, there are a lot more feature available but any feature not added to the file gets it's default value. More on this can be seen in https://www.playframework.com/documentation/2.6.x/Configuration.

```
# Application ID
app {
  name = "msb-router"
  version = "1.0.0"
  address = "127.0.0.1"
  port = 50001
  akka.port = 41001
}

# IDE
play.editor = "http://localhost:63342/api/file/?file=%s&line=%s"

# Modules
play.modules {
  enabled += "modules.Module"
}

# Play HTTP settings
play.http {
  # ErrorHandler
  errorHandler = "services.ErrorHandler"

  # filters.Filters
  filters = "filters.Filters"

  # Secret key
  secret {
    key = "changeme"
  }
}

# Play server config
play.server {
  dir = ${?user.dir}

  provider = "play.core.server.AkkaHttpServerProvider"

  # HTTP configuration
  http {
    port = disabled
    # The HTTP port of the server. Use a value of "disabled" if the server
shouldn't bind an HTTP port.
    port = ${app.port}
    # The interface address to bind to.
    address = "0.0.0.0"
    # The idle timeout for an open connection after which it will be closed. Set
to null to disable the timeout
    idleTimeout = 60s
  }

  # HTTPS configuration
  https {
    port = 50001
    port = ${?https.port}
    # The interface address to bind to.
    address = "0.0.0.0"
    # The idle timeout for an open connection after which it will be closed. Set
to null to disable the timeout
    idleTimeout = ${play.server.http.idleTimeout}
  }

  # How long a request takes until it times out
  requestTimeout = 40s

  # The path to the process id file created by the server when it runs.
  pidfile.path = ${play.server.dir}/RUNNING_PID

  websocket {
    frame.maxLength = 64k
  }
}


## WS (HTTP Client)
play.ws {
  ## WS SSL
  ssl {
    trustManager = {
      stores = [
        {type: "JKS", path: "./conf/mhp.jks", password: "xxx"}
        {path: ${java.home}/lib/security/cacerts, password: "changeit"} #
Fallback to default JSSE trust store
      ]
    }

    #loose.acceptAnyCertificate = true

    debug {
      # Turn on all debugging
      all = false
      # Turn on ssl debugging
      ssl = false
      # Turn certpath debugging on
      certpath = false
      # Turn ocsp debugging on
      ocsp = false
      # Enable per-record tracing
      record = false
      # hex dump of record plaintext, requires record to be true
      plaintext = false
      # print raw SSL/TLS packets, requires record to be true
      packet = false
      # Print each handshake message
      handshake = false
      # Print hex dump of each handshake message, requires handshake to be true
      data = false
      # Enable verbose handshake message printing, requires handshake to be true
      verbose = false
      # Print key generation data
      keygen = false
      # Print session activity
      session = false
      # Print default SSL initialization
      defaultctx = false
      # Print SSLContext tracing
      sslctx = false
      # Print session cache tracing
      sessioncache = false
      # Print key manager tracing
      keymanager = false
      # Print trust manager tracing
      trustmanager = false
      # Turn pluggability debugging on
      pluggability = false
    }
  }
}


## filters.Filters
play.filters {

  ## CORS filter configuration
  # https://www.playframework.com/documentation/latest/CorsFilter
  cors {
  }

  ## CSRF Filter
  # https://www.playframework.com/documentation/latest/ScalaCsrf#Applying-a-
global-CSRF-filter
  csrf {
  }

  ## Security headers filter configuration
  headers {
    contentSecurityPolicy = "script-src 'self' 'unsafe-inline' clef.io
jquery.min.js cdnjs.cloudflare.com;"
  }

  ## Allowed hosts filter configuration
  hosts.allowed = ["."]
}

## Mailer
#
play.mailer {
  # SMTP serveur, example : smtp.gmail.com
  host = smtp.gmail.com

  # Mail Port, example : 465, 587 or 25
  port = 587

  # Mail Auth User, example : user@gmail.com
  user = "xxx@gmail.com"

  # Mail Auth Password
  password = xxx

  # Mail SSL : true or false
  ssl = false
  tls = true

  # Will only log all the email properties instead of sending an email
  mock = false

  # Mail user from
  from = "PbDevStage <xxx@gmail.com>"
  reply = "No reply <noreply@gmail.com>"
}


play.assets {
  path = "/public"
  urlPrefix = "/assets"
}

include "application.akka.conf"
include "application.playguard.conf"
```

Code 19 – Plays main configuration file example

The next example is from the configuration file of the Akka Cluster.

```
##
app {
  akka {
    router {
      auth.active = true
      ...
    }

    monitor {
      cluster-events = true
      cluster-metrics = false
    }
  }
}

## Akka
play.akka.actor-system = "mhp"

$
akka {
   actor {
    provider = "cluster"
    deployment {
      # Auth Router
      /routerActorSupervisor/router/authRouter {
        #router = round-robin-group | adaptive-group
        # Router type provided by metrics extension.
        router = cluster-metrics-adaptive-group
        # metrics-selector = heap
        # metrics-selector = load
        # metrics-selector = cpu
        metrics-selector = mix

        routees.paths = ["/user/authWorker"]
        #max-total-nr-of-instances =
        cluster {
          enabled = on
          use-role = auth
          allow-local-routees = off
        }
      }
    }
  }


  cluster {
    #min-nr-of-members = 2

    seed-nodes = [
      "akka.tcp://msb@127.0.0.1:41001",
      "akka.tcp://msb@127.0.0.1:41011"]

    roles = [router]

    # how long to wait for one of the seed nodes to reply to initial join request
    seed-node-timeout = 10s
    # If a join request fails it will be retried after this period. Disable join
retry by specifying "off".
    retry-unsuccessful-join-after = 15s

    # disable in production
    auto-down-unreachable-after = 10s

    # disable legacy metrics in akka-cluster, since it is still enabled in akka-
cluster by default
    metrics {
      enabled = off
      native-library-extract-folder = ${user.dir}/target/native

      # Metrics collector actor.
      collector {
        # Enable or disable metrics collector for load-balancing nodes.
        enabled = on
        provider = ""
        # Try all 3 available collector providers, or else fail on the configured
custom collector provider.
        fallback = true
        # How often metrics are sampled on a node.
        sample-interval = 5s
        # How often a node publishes metrics information to the other nodes in
the cluster.
        # Shorter interval will publish the metrics gossip more often. (default
3s)
        gossip-interval = 5s
        # How quickly the exponential weighting of past data is decayed compared
to
        moving-average-half-life = 12s
      }
    }
  }

  extensions = ["akka.cluster.metrics.ClusterMetricsExtension"]

  remote {
    enabled-transports = ["akka.remote.netty.tcp"]
    netty.tcp {
      hostname = ${app.address}
      port = ${app.akka.port}
    }
    # After failed to establish an outbound connection, the remoting will mark
the
    # address as failed.
    retry-gate-closed-for = 10 s
    # Settings for the failure detector to monitor connections.
    transport-failure-detector {

      # FQCN of the failure detector implementation.
      implementation-class = "akka.remote.DeadlineFailureDetector"

      # How often keep-alive heartbeat messages should be sent to each
connection.
      heartbeat-interval = 4 s

      # Number of potentially lost/delayed heartbeats that will be
      # accepted before considering it to be an anomaly.
      acceptable-heartbeat-pause = 120 s
    }

    watch-failure-detector {
      # FQCN of the failure detector implementation.
      implementation-class = "akka.remote.PhiAccrualFailureDetector"

      # How often keep-alive heartbeat messages should be sent to each
connection.
      heartbeat-interval = 3 s

      # Defines the failure detector threshold.
      threshold = 10.0

      # Number of the samples of inter-heartbeat arrival times to adaptively
      # calculate the failure timeout for connections.
      max-sample-size = 200

      # Minimum standard deviation to use for the normal distribution in
      # AccrualFailureDetector.
      min-std-deviation = 100 ms

      # Number of potentially lost/delayed heartbeats that will be
      # accepted before considering it to be an anomaly.
      acceptable-heartbeat-pause = 10 s

      # How often to check for nodes marked as unreachable by the failure
detector
      unreachable-nodes-reaper-interval = 3s

      # After the heartbeat request has been sent the first failure detection
      # will start after this period, even though no heartbeat message has
      # been received.
      expected-response-after = 1 s
    }
  }
}
```

Code 20 – Plays Akka configuration file example

6.6. Documentation File Example

Confidential information was removed the file.

```
swagger: '2.0'

info:

  description: |
    API documentation for the Authentication service

    Todas as mensagens são enviadas com codificação UTF-8.

    As mensagens são <i>case-sensitive</i> e todos os campos são construídos em
<i>Lower Camel Case</i>.

    Notas:
      * Os exemplos abaixo são meramente para efeitos de demonstração e não
correspondem a dados reais (não usar em testes)
      * Para mais informação consultar manual técnico.

    * General Error Codes: https://xxx.pt:50001/api/docs/error-codes

  title: Serviço de Autenticação/Autorização

  version: 1.0.0

  contact:
    name: LTS
    url: http://xxx.pt
    email: xxx@xxx.pt

  license:
    name: Comercial
    url: http://xxx.pt

host: 'xxx.pt:50001'

basePath: /api/auth

tags:
  - name: auth
    description: Documentação para o serviço de Autenticação
    externalDocs:
      description: Mais informações
      url: https://xxx.pt:50001/api/docs/auth

schemes:
  - https

paths:

  /signIn:
    post:
      tags:
        - auth
      summary: Autentica uma conta de utilizador
      description: 'As credenciais necessitam de ser requisitadas préviamente à
```

```
LTS'
      operationId: ''
      consumes:
        - application/json
      produces:
        - application/json
      parameters:
        - in: body
          name: body
          description: Objecto a adicionar ao pedido
          required: true
          schema:
            $ref: '#/definitions/signIn'
      responses:
        '200':
          description: 'Ok'
          schema:
            $ref: '#/definitions/status'
        '401':
          description: 'https://xxx.pt:50001/api/docs/error-codes'
        '405':
          description: 'https://xxx.pt:50001/api/docs/error-codes'
        '406':
          description: 'https://xxx.pt:50001/api/docs/error-codes'

securityDefinitions:
  bearer:
    type: apiKey
    name: x-auth-token
    in: header

definitions:

  signIn:
    type: object
    description: Objecto que identifica o tipo de mensagem.
    required:
      - email
      - password
      - rememberMe
    properties:
      email:
        description: O email to utilizador
        type: string
        example: john
      password:
        description: A password do utilizador
        type: string
        example: doe
      rememberMe:
        description: Se o sistema aceita o mesmo token por um periodo extendido de
tempo. Útil se forem realizados vários pedidos sequenciais.
        type: boolean
        example: false

  status:
    type: object
    description: Detalhe da mensagem de Estado
    required:
      - status
    properties:
      status:
        required:
          - i
          - s
          - v
        properties:
          i:
            description: (Id) Código da resposta
            type: string
            maxLength: 6
            example: 200
          s:
            description: (Status) Estado da mensagem enviada (ver Mensagens de
Retorno)
            type: string
            maxLength: 200
            example: Ok
          v:
            description: (Version) Versão do inicial do serviço a partil do qual a
mensagem de retorno foi implementada e/ou alterada.
            type: string
            maxLength: 20
            example: 1.0.0
          t:
            description: (Token) Identificador do pedido original, se aplicavel.
            type: string
            maxLength: 36
            example: XXX
          ai:
            description: Uso interno. Enviar juntamente com os pedidos de suporte.
            type: string
            maxLength: 200
            example: XXX
```

Code 21 – Authorization documentation YAML file

6.6. Logback Configuration File Example

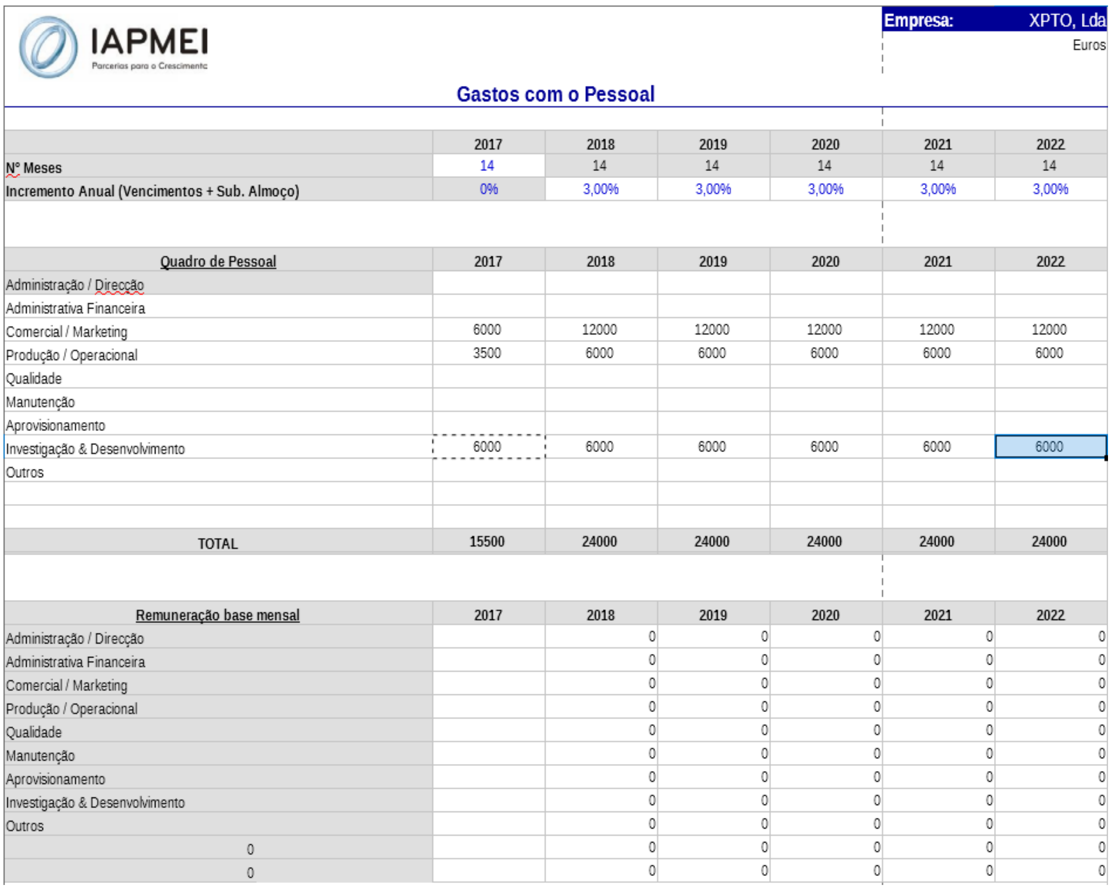
```
<configuration>
    <property name="LOG_NAME" value="msb-auth"/>

    <!-- Setting context name helps distinguish between different applications. Can
be used in the patterns (%contextName) -->
    <contextName>${LOG_NAME}</contextName>

    <conversionRule conversionWord="coloredLevel"
converterClass="play.api.libs.logback.ColoredLevel"/>

    <!-- FILE -->
    <appender name="FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">
        <file>${application.home:-.}/logs/${LOG_NAME}.log</file>
        <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
            <!-- Daily rollover with compression -->
            <fileNamePattern>${application.home:-.}/logs/${LOG_NAME}-log-%d{yyyy-
MM-dd}.gz</fileNamePattern>
            <!-- keep 30 days worth of history -->
            <maxHistory>60</maxHistory>
        </rollingPolicy>
        <encoder>
            <pattern>%date{yyyy-MM-dd HH:mm:ss:SSS} %.-3level %message [%logger in
%thread] %n%xException</pattern>
        </encoder>
    </appender>

    <!-- ACCESS_FILE -->
    <appender name="ACCESS_FILE"
class="ch.qos.logback.core.rolling.RollingFileAppender">
        <file>${application.home:-.}/logs/access/${LOG_NAME}-access.log</file>
        <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
            <!-- daily rollover with compression -->
            <fileNamePattern>${application.home:-.}/logs/${LOG_NAME}-access-log-
%d{yyyy-MM-dd}.gz</fileNamePattern>
            <!-- keep 1 week worth of history -->
            <maxHistory>15</maxHistory>
        </rollingPolicy>
        <encoder>
            <pattern>%date{yyyy-MM-dd HH:mm:ss ZZZZ} %message%n</pattern>
        </encoder>
    </appender>

    <!-- STDOUT -->
    <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
        <encoder>
            <pattern>%date{HH:mm:ss:SSS} %highlight(%.-3level) %message
[%cyan(%logger{50} in %thread])
                %n%xException{50}
            </pattern>
        </encoder>
    </appender>

    <!-- EMAIL -->
    <appender name="EMAIL" class="ch.qos.logback.classic.net.SMTPAppender">
```

```
        <smtpHost>smtp.gmail.com</smtpHost>
        <smtpPort>587</smtpPort>
        <STARTTLS>true</STARTTLS>
        <username>autobot@logitools.pt</username>
        <password>skrfwlppjbeutfss</password>
        <to>hugotigre@logitools.pt</to> <!-- additional destinations are possible
-->
        <from>autobot@logitools.pt</from>
        <subject>${LOG_NAME}: %logger{20} - %m</subject>
        <!--<layout class="ch.qos.logback.classic.PatternLayout">
            <pattern>%date %-5level %logger{35} - %message%n</pattern>
        </layout>-->
        <layout class="ch.qos.logback.classic.html.HTMLLayout">
            <pattern>%relative%thread%mdc%(%.-3level)%logger%msg</pattern>
        </layout>
    </appender>


    <!--
        Wrap the appender(s) in async appender(s)
    -->
    <appender name="ASYNCFILE" class="ch.qos.logback.classic.AsyncAppender">
        <queueSize>500</queueSize>
        <!--<discardingThreshold>0</discardingThreshold>-->
        <maxFlushTime>30000</maxFlushTime>
        <appender-ref ref="FILE"/>
    </appender>

    <!-- -->
    <appender name="ASYNCSTDOUT" class="ch.qos.logback.classic.AsyncAppender">
        <maxFlushTime>30000</maxFlushTime>
        <appender-ref ref="STDOUT"/>
    </appender>


    <appender name="ASYNCACCESSFILE" class="ch.qos.logback.classic.AsyncAppender">
        <maxFlushTime>30000</maxFlushTime>
        <appender-ref ref="ACCESS_FILE"/>
    </appender>

    <appender name="ASYNCEMAIL" class="ch.qos.logback.classic.AsyncAppender">
        <maxFlushTime>30000</maxFlushTime>
        <appender-ref ref="EMAIL"/>
    </appender>

    <!--
        Loggers
    -->
    <!-- basic play loggers -->
    <logger name="play.api" level="INFO"/>
    <logger name="application" level="INFO"/>
    <logger name="ch.qos.logback" level="WARN"/>
    <logger name="com.google.inject" level="WARN"/>
    <logger name="net.sf.ehcache" level="WARN"/>
    <logger name="org.asynchttpclient.netty" level="INFO"/>
    <logger name="io.netty" level="INFO"/>
    <logger name="org.jboss" level="INFO"/>

    <!-- akka loggers -->
    <logger name="akka" level="INFO"/>
    <!-- other loggers -->
    <logger name="javax" level="INFO"/>
    <logger name="sun.security" level="INFO"/>
    <!-- MongoDB -->
    <logger name="com.mohiva" level="DEBUG"/>
    <logger name="reactivemongo" level="WARN"/>

    <!-- additivity=false ensures access log data only goes to the access log -->
    <logger name="access" level="INFO" additivity="false">
        <appender-ref ref="ASYNCACCESSFILE"/>
    </logger>


    <!--
        Add appender(s) to root
    -->
    <root level="INFO">
        <appender-ref ref="ASYNCFILE"/>
        <!--<appender-ref ref="ASYNCSTDOUT"/>-->
        <appender-ref ref="ASYNCEMAIL"/>
    </root>

    <!-- Gracefully terminate logback when JVM terminates -->
    <shutdownHook class="ch.qos.logback.core.hook.DelayingShutdownHook"/>
</configuration>
```

Code 22 – Logback configuration file example

6.7. Cost of Development

The following figures give an overview of the main costs related to the development and implementation of the platform.



IAPMEI
Parcerias para o Crescimento

Empresa:	XPTO, Lda
	Euros

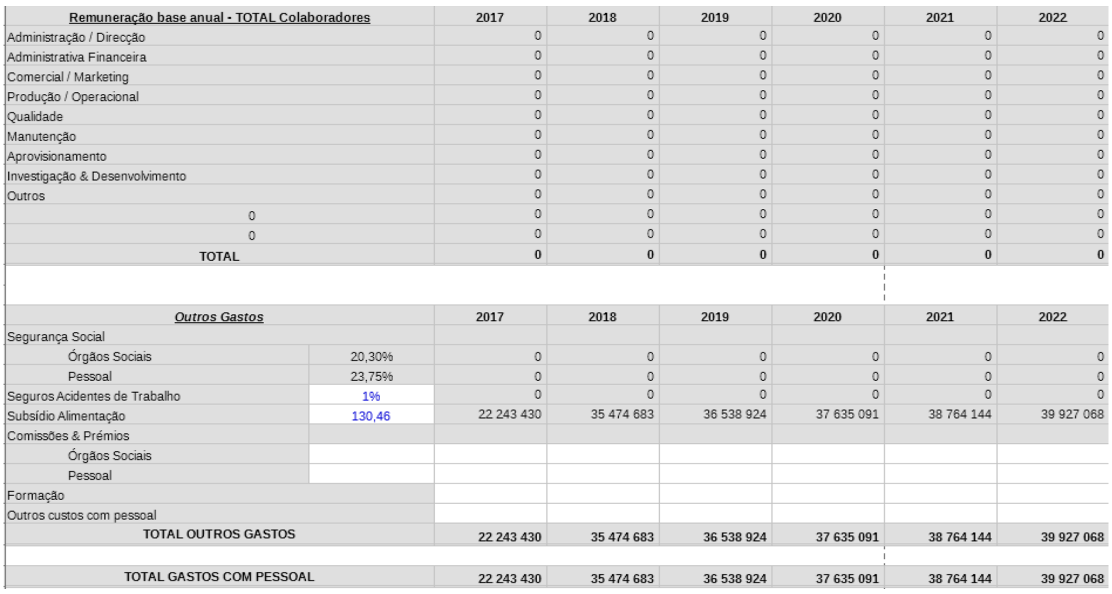
Gastos com o Pessoal

	2017	2018	2019	2020	2021	2022
Nº Meses	14	14	14	14	14	14
Incremento Anual (Vencimentos + Sub. Almoço)	0%	3,00%	3,00%	3,00%	3,00%	3,00%

Quadro de Pessoal	2017	2018	2019	2020	2021	2022
Administração / Direcção						
Administrativa Financeira						
Comercial / Marketing	6000	12000	12000	12000	12000	12000
Produção / Operacional	3500	6000	6000	6000	6000	6000
Qualidade						
Manutenção						
Aprovisionamento						
Investigação & Desenvolvimento	6000	6000	6000	6000	6000	6000
Outros						
TOTAL	15500	24000	24000	24000	24000	24000

Remuneração base mensal	2017	2018	2019	2020	2021	2022
Administração / Direcção		0	0	0	0	0
Administrativa Financeira		0	0	0	0	0
Comercial / Marketing		0	0	0	0	0
Produção / Operacional		0	0	0	0	0
Qualidade		0	0	0	0	0
Manutenção		0	0	0	0	0
Aprovisionamento		0	0	0	0	0
Investigação & Desenvolvimento		0	0	0	0	0
Outros		0	0	0	0	0
0		0	0	0	0	0
0		0	0	0	0	0

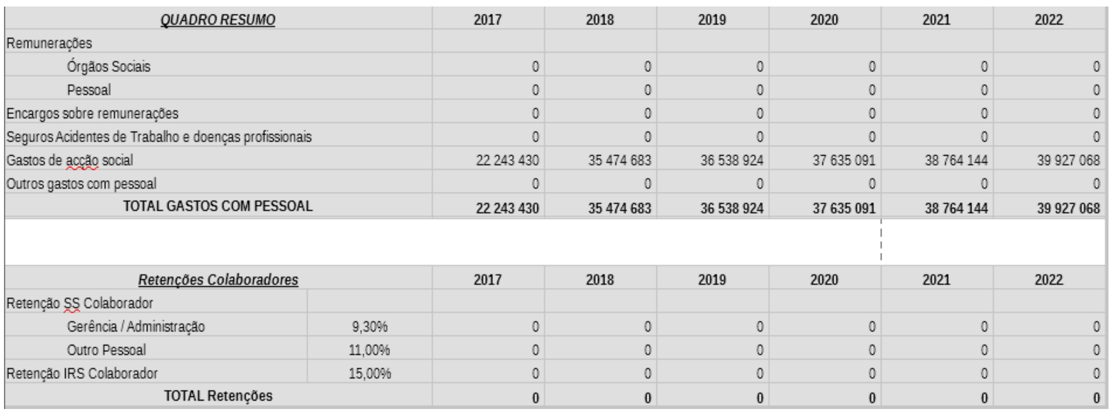
Figure 37: Development investment 1



Remuneração base anual - TOTAL Colaboradores	2017	2018	2019	2020	2021	2022
Administração / Direcção	0	0	0	0	0	0
Administrativa Financeira	0	0	0	0	0	0
Comercial / Marketing	0	0	0	0	0	0
Produção / Operacional	0	0	0	0	0	0
Qualidade	0	0	0	0	0	0
Manutenção	0	0	0	0	0	0
Aprovisionamento	0	0	0	0	0	0
Investigação & Desenvolvimento	0	0	0	0	0	0
Outros	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
TOTAL	**0**	**0**	**0**	**0**	**0**	**0**

Outros Gastos		2017	2018	2019	2020	2021	2022
Segurança Social							
Órgãos Sociais	20,30%	0	0	0	0	0	0
Pessoal	23,75%	0	0	0	0	0	0
Seguros Acidentes de Trabalho	1%	0	0	0	0	0	0
Subsídio Alimentação	130,46	22 243 430	35 474 683	36 538 924	37 635 091	38 764 144	39 927 068
Comissões & Prémios							
Órgãos Sociais							
Pessoal							
Formação							
Outros custos com pessoal							
TOTAL OUTROS GASTOS		**22 243 430**	**35 474 683**	**36 538 924**	**37 635 091**	**38 764 144**	**39 927 068**
TOTAL GASTOS COM PESSOAL		**22 243 430**	**35 474 683**	**36 538 924**	**37 635 091**	**38 764 144**	**39 927 068**

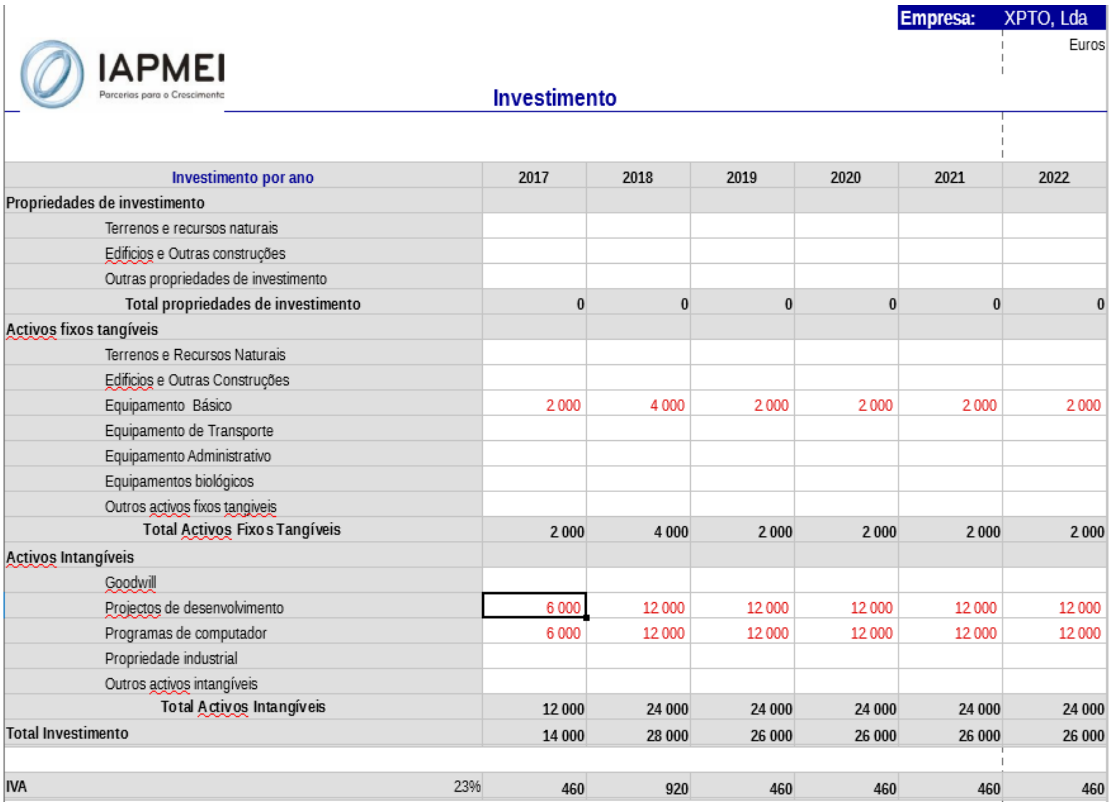
Figure 38: Development investment 2



QUADRO RESUMO	2017	2018	2019	2020	2021	2022
Remunerações						
Órgãos Sociais	0	0	0	0	0	0
Pessoal	0	0	0	0	0	0
Encargos sobre remunerações	0	0	0	0	0	0
Seguros Acidentes de Trabalho e doenças profissionais	0	0	0	0	0	0
Gastos de acção social	22 243 430	35 474 683	36 538 924	37 635 091	38 764 144	39 927 068
Outros gastos com pessoal	0	0	0	0	0	0
TOTAL GASTOS COM PESSOAL	**22 243 430**	**35 474 683**	**36 538 924**	**37 635 091**	**38 764 144**	**39 927 068**

Retenções Colaboradores		2017	2018	2019	2020	2021	2022
Retenção SS Colaborador							
Gerência / Administração	9,30%	0	0	0	0	0	0
Outro Pessoal	11,00%	0	0	0	0	0	0
Retenção IRS Colaborador	15,00%	0	0	0	0	0	0
TOTAL Retenções		**0**	**0**	**0**	**0**	**0**	**0**

Figure 39: Development investment 3



Empresa: XPTO, Lda

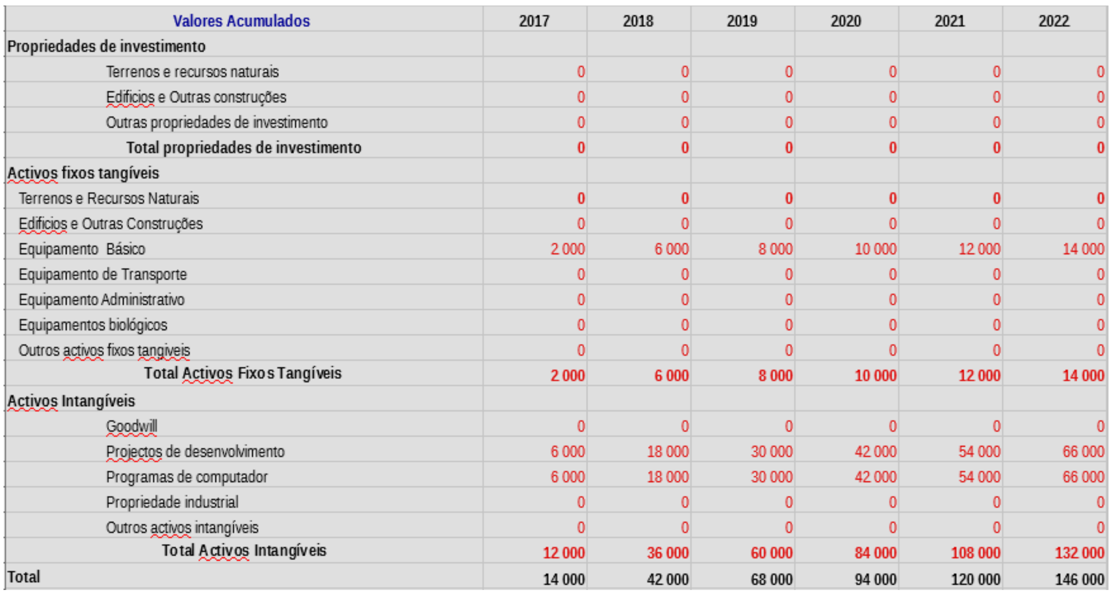
Euros

IAPMEI Parcerias para o Crescimento

Investimento

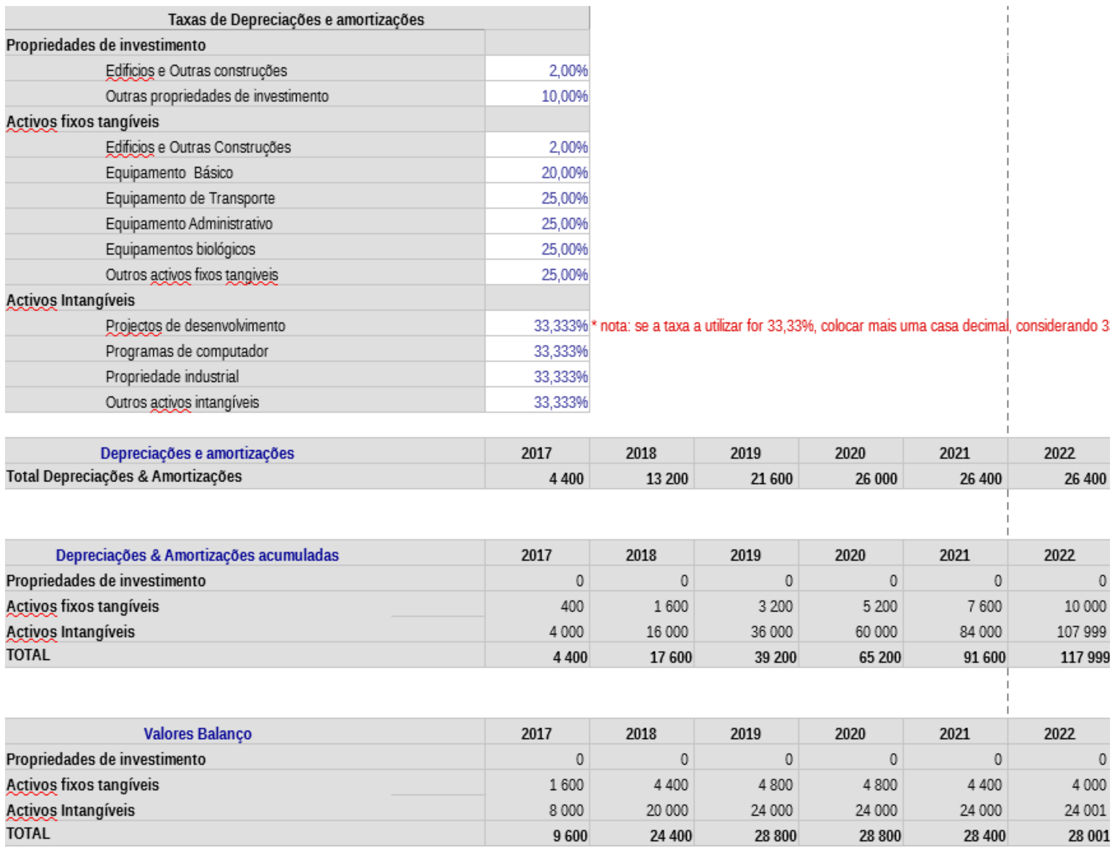
Investimento por ano		2017	2018	2019	2020	2021	2022
Propriedades de investimento							
Terrenos e recursos naturais							
Edifícios e Outras construções							
Outras propriedades de investimento							
Total propriedades de investimento		**0**	**0**	**0**	**0**	**0**	**0**
Activos fixos tangíveis							
Terrenos e Recursos Naturais							
Edifícios e Outras Construções							
Equipamento Básico		2 000	4 000	2 000	2 000	2 000	2 000
Equipamento de Transporte							
Equipamento Administrativo							
Equipamentos biológicos							
Outros activos fixos tangíveis							
Total Activos Fixos Tangíveis		**2 000**	**4 000**	**2 000**	**2 000**	**2 000**	**2 000**
Activos Intangíveis							
Goodwill							
Projectos de desenvolvimento		6 000	12 000	12 000	12 000	12 000	12 000
Programas de computador		6 000	12 000	12 000	12 000	12 000	12 000
Propriedade industrial							
Outros activos intangíveis							
Total Activos Intangíveis		**12 000**	**24 000**	**24 000**	**24 000**	**24 000**	**24 000**
Total Investimento		**14 000**	**28 000**	**26 000**	**26 000**	**26 000**	**26 000**
IVA	23%	**460**	**920**	**460**	**460**	**460**	**460**

Figure 40: Development investment 4



Valores Acumulados	2017	2018	2019	2020	2021	2022
Propriedades de investimento						
Terrenos e recursos naturais	0	0	0	0	0	0
Edifícios e Outras construções	0	0	0	0	0	0
Outras propriedades de investimento	0	0	0	0	0	0
Total propriedades de investimento	**0**	**0**	**0**	**0**	**0**	**0**
Activos fixos tangíveis						
Terrenos e Recursos Naturais	**0**	**0**	**0**	**0**	**0**	**0**
Edifícios e Outras Construções	0	0	0	0	0	0
Equipamento Básico	2 000	6 000	8 000	10 000	12 000	14 000
Equipamento de Transporte	0	0	0	0	0	0
Equipamento Administrativo	0	0	0	0	0	0
Equipamentos biológicos	0	0	0	0	0	0
Outros activos fixos tangíveis	0	0	0	0	0	0
Total Activos Fixos Tangíveis	**2 000**	**6 000**	**8 000**	**10 000**	**12 000**	**14 000**
Activos Intangíveis						
Goodwill	0	0	0	0	0	0
Projectos de desenvolvimento	6 000	18 000	30 000	42 000	54 000	66 000
Programas de computador	6 000	18 000	30 000	42 000	54 000	66 000
Propriedade industrial	0	0	0	0	0	0
Outros activos intangíveis	0	0	0	0	0	0
Total Activos Intangíveis	**12 000**	**36 000**	**60 000**	**84 000**	**108 000**	**132 000**
Total	**14 000**	**42 000**	**68 000**	**94 000**	**120 000**	**146 000**

Figure 41: Development investment 5



Taxas de Depreciações e amortizações	
Propriedades de investimento	
Edifícios e Outras construções	2,00%
Outras propriedades de investimento	10,00%
Activos fixos tangíveis	
Edifícios e Outras Construções	2,00%
Equipamento Básico	20,00%
Equipamento de Transporte	25,00%
Equipamento Administrativo	25,00%
Equipamentos biológicos	25,00%
Outros activos fixos tangíveis	25,00%
Activos Intangíveis	
Projectos de desenvolvimento	33,333%
Programas de computador	33,333%
Propriedade industrial	33,333%
Outros activos intangíveis	33,333%

* nota: se a taxa a utilizar for 33,33%, colocar mais uma casa decimal, considerando 3

Depreciações e amortizações	2017	2018	2019	2020	2021	2022
Total Depreciações & Amortizações	**4 400**	**13 200**	**21 600**	**26 000**	**26 400**	**26 400**

Depreciações & Amortizações acumuladas	2017	2018	2019	2020	2021	2022
Propriedades de investimento	0	0	0	0	0	0
Activos fixos tangíveis	400	1 600	3 200	5 200	7 600	10 000
Activos Intangíveis	4 000	16 000	36 000	60 000	84 000	107 999
TOTAL	**4 400**	**17 600**	**39 200**	**65 200**	**91 600**	**117 999**

Valores Balanço	2017	2018	2019	2020	2021	2022
Propriedades de investimento	0	0	0	0	0	0
Activos fixos tangíveis	1 600	4 400	4 800	4 800	4 400	4 000
Activos Intangíveis	8 000	20 000	24 000	24 000	24 000	24 001
TOTAL	**9 600**	**24 400**	**28 800**	**28 800**	**28 400**	**28 001**

Figure 42: Development investment 6