1 2 9 0

UNIVERSIDADE D
COIMBRA

João David Marques Santos Ribeiro

A DASHBOARD FOR DECISION SUPPORT IN SELF-ADAPTIVE CLOUD APPLICATIONS

Dissertation in the context of the Master in Informatics Engineering, specialization in Software Engineering, advised by Professor Nuno Antunes and José Pereira and presented to the Department of Informatics Engineering of the Faculty of Sciences and Technology of the University of Coimbra.

July of 2022

1 2 9 0

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE Đ
COIMBRA

DEPARTMENT OF INFORMATICS ENGINEERING

João David Marques Santos Ribeiro

A Dashboard for Decision Support in Self-Adaptive Cloud Applications

Dissertation in the context of the Master in Informatics Engineering, specialization in Software Engineering, advised by Prof. Nuno Antunes and José Pereira and presented to the Department of Informatics Engineering of the Faculty of Sciences and Technology of the University of Coimbra.

July 2022

The work presented in this dissertation was carried out within the Software and Systems Engineering (SSE) group of the Centre for Informatics and Systems of the University of Coimbra (CISUC).

This work is partially supported by the project **TalkConnect - Voice Architecture over Distributed Network** (reference `POCI-01-0247-FEDER-039676`) financed by Fundo Europeu de Desenvolvimento Regional (FEDER) through COMPETE 2020, from Portugal 2020.

This work has been supervised by Nuno Manuel dos Santos Antunes and José Alexandre D'Abruzzo Pereira, Professors at the Department of Informatics Engineering of the Faculty of Sciences and Technology of the University of Coimbra.

COMPETE 2020 PORTUGAL 2020 UNIÃO EUROPEIA Fundo Europeu de Desenvolvimento Regional

Acknowledgements

I would like to thank professor Nuno Antunes and José Pereira for all the guidance, orientation, and knowledge provided in the course of this dissertation's work. Without them, I would not have learned so many things as I did while working and being guided and supervised by them.

I also would like to thank professors Jorge Granjal and Rui Paiva for all the feedback provided in the mid-term presentation which was extremely useful for guiding the work in the second semester.

Also, I would like to thank Miguel Teixeira, Henrique Silva, Paulo Gonçalves, Pedro Almeida, Carlos Santos, and Francisco Guerra for the amazing group we created where we interchanged opinions, discussed, learned from each other, and worked and laughed together.

I want to thank my girlfriend, Inês Bernardes, for all the support, motivation, patience, and love that were very important at this phase of my life.

Above all, I am grateful to my parents for all the unconditional love, work and dedication they had that allowed me to be here, at this important phase of my life, writing this document.

Abstract

Cloud computing has become so popular and widely adopted that plenty of applications, or parts of it, are hosted in a cloud environment. Furthermore, with emerging technologies like containers that improve the service offered by clouds, systems continue to grow and become larger, more fragmented, and more complex. That complexity requires constant monitoring and adaptation to keep up with demand, and maintain or improve performance without increasing costs unnecessarily. Also, if a system can self-adapt, management activity becomes easier as adaptations no longer require supervisors to manually perform them.

Although there are some interesting monitoring solutions, they do not offer automation of the adaptability part. Thus, self-adaptive enabling tools become much more attractive as they offer automation of the whole systems management activity, from monitoring to adaptations. Among self-adaptive enabling tools, there are some disadvantages such as environmental restrictions to cloud providers, the inability to manage more than one system, and the nonexistence of a complete control loop capable of integrating with any type of system. TMA (Trustworthiness Monitoring & Assessment Framework) is a monitoring and management platform that implements a complete control loop capable of providing any system with self-adaptive capabilities. It dispatches adaptations on systems based on quality models, which are weighted trees of metrics representing the requirements of systems. Also, it can manage more than a single system.

In this work it was developed a dashboard for TMA, focusing on functionalities related to systems monitoring such as the creation and management of charts, metrics values simulations, and the creation of metrics and quality models. To support this, TMA's API component had to be extended and the Analyze component automated. Additionally, features for managing adaptation rules were created so, TMA's Planning component had its architecture changed and its code extended and altered.

The dashboard was implemented upon React's framework, using Semantic UI for styling, while the TMA components were reprogrammed in the same language they were written. The choice of development frameworks for the dashboard was based on multiple factors such as experience, responsiveness, and simplicity. Also, as Maintainability and Usability are attributes the product should have, React was chosen as it suits both properties. Furthermore, it is a trending technology, which means there is a lot of support that should ease the development activity.

Keywords

Cloud applications, *self-adaptive systems*, *TMA*, *decision support*, *interface*, *monitoring*

Resumo

A computação em nuvem tornou-se tão popular e amplamente adoptada que muitas aplicações, ou partes delas, estão alojadas num ambiente cloud. Além disso, com tecnologias emergentes como os containers que melhoram o serviço oferecido pelas clouds, os sistemas continuam a crescer e a tornar-se maiores, mais fragmentados e mais complexos. Essa complexidade requer constante monitorização e adaptação para lidar com a carga, e manter ou melhorar o desempenho sem aumentar desnecessariamente os custos. Além disso, se um sistema se conseguir auto-adaptar, a tarefa de gestão é facilitada, uma vez que deixa de ser necessário que os supervisores executem as adaptações manualmente.

Embora existam algumas soluções de monitorização interessantes, elas não oferecem automatização da parte da adaptação. Assim, ferramentas que permitem auto-adaptação tornam-se muito mais atractivas, uma vez que oferecem automatização de toda a actividade de gestão de sistemas, desde monitorização a adaptações. Entre estas ferramentas existem algumas desvantagens, tais como a restrição ambiental a fornecedores cloud, incapacidade de gerir mais do que um sistema e inexistência de um ciclo de controlo completo capaz de se integrar com qualquer tipo de sistema. O TMA (Trustworthiness Monitoring & Assessment Framework) é uma plataforma de monitorização e gestão que permite auto-adaptação em sistemas. Ele executa adaptações baseando-se em modelos de qualidade, que são árvores de métricas ponderadas que representam os requisitos dos sistemas. Além disso, o TMA pode gerir mais do que um único sistema.

Neste trabalho foi desenvolvida uma dashboard para o TMA, focada em funcionalidades de monitorização de sistemas como a criação e gestão de gráficos, simulações de valores de métricas, e criação de métricas e modelos de qualidade. Para isso, a componente API do TMA teve de ser extendida e a componente Analyze de ser automatizada. Adicionalmente, funcionalidades para a gestão das regras de adaptação foram criadas, por isso a arquitetura da componente Planning do TMA foi alterada e o seu código alargado e alterado.

A dashboard foi implementada utilizando a framework React, e a framework Semantic UI para estilização, enquanto que as componentes do TMA foram reprogramadas na mesma linguagem em que foram escritas. A escolha das frameworks foi baseada em experiência, responsividade e simplicidade. Além disso, a manutenção e a usabilidade são atributos que o produto deve possuir, daí a escolha do React por ser adequado a ambos. Sobre isso, é uma tecnologia popular, o que significa que há muito suporte e, portanto, a actividade de desenvolvimento deverá ser facilitada.

Palavras-Chave

aplicações Cloud, sistemas auto-adaptativos, TMA, suporte à decisão, interface, monitorização

List of Publications

The work of this dissertation resulted in the following publication that is submitted for reviewing in a conference:

- João Ribeiro, José Pereira, and Nuno Antunes. "An Experimental Study of Elasticity in Kubernetes HPA for Microservice Applications", 13º INForum, Guarda, Portugal, 8-9 September, 2022.
  - ***Abstract:*** *Microservice applications are usually deployed in a cloud environment, which has a cost per resource used in an Infrastructure as a Service (IaaS) model. Hence, it is important to allocate only the resources needed to provide a given quality of service. Container management systems (e.g., Kubernetes) can be used to deploy microservice applications. Although cloud providers have autoscaling services at the instance level, either they do not have them at the container level, or those services are not easy to use. In this study, we evaluate the elasticity of Kubernetes Horizontal Pod Autoscaler (HPA), which adds or removes containers (pods) of a service based on the mean CPU usage of the deployed ones. To do this assessment, the microservice application TeaStore is used, and we created a workload divided into two phases with diverse operational profiles. Results show that using the self-adaptive autoscaler HPA increases the throughput by 5%, either when the load exceeds the threshold that the default configuration can handle or not. Also, as the HPA scales only the overloaded services, the resources spent for the throughput speedup show efficiency gains.*

Contents

Acronyms

API Application Programming Interface.

BLOB Binary Large Object.

CISUC Centre for Informatics and Systems of the University of Coimbra.

CLI Command Line Interface.

CPU Central Processing Unit.

CSS Cascading Style Sheet.

DOM Document Object Model.

HPA Horizontal Pod AutoScaler.

HTTP Hypertext Transfer Protocol.

IaaS Infrastructure-As-A-Service.

IP Internet Protocol.

IT Information Technology.

JSX JavaScript Syntax Extension.

MVVM Model-View-ViewModel.

PaaS Platform-As-A-Service.

PDF Portable Document Format.

RAM Random Access Memory.

REST Representational State Transfer.

SaaS Software-As-A-Service.

SEO Search Engine Optimization.

SPA Single Page Application.

SSE Software and Systems Engineering.

TMA Trustworthiness Monitoring & Assessment Framework.

UC	University of Coimbra.
UI	User Interface.
VMs	Virtual Machines.
YAML	YAML Ain't Markup Language.

List of Figures

List of Tables

Chapter 1

Introduction

Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction [1].

There are three main cloud service types: *Software-As-A-Service (SaaS)*, *Platform-As-A-Service (PaaS)* and *Infrastructure-As-A-Service (IaaS)* [2]: **SaaS** is related to cloud consumers releasing their applications on the cloud, which can then be accessed from different networks of application users. **PaaS** is about providing a development platform which allows cloud consumers to develop cloud services and applications directly on the cloud. **IaaS** is a service model that allows cloud consumers to directly use Information Technology (IT) infrastructure (e.g., processing, storage, and networks) provided in the cloud.

Computer science has progressed and new technologies emerged. *Virtualization* is one that is highly leveraged by Cloud providers, used to emulate different environments, Virtual Machines (VMs), under a single physical machine [3]. In IaaS cloud service model, resource virtualization is the main concept. It allows the user to have his own guest operating system on top of the infrastructure provided by the cloud provider [4]. Thus, it becomes possible to maximize the usage of the hardware resources through machine sharing and consequently, cloud providers can boost their revenues and lower prices. As the same machine can have different, separated, isolated, and hardware customized environments, customers may be served with the same machine without having a perception of it.

Nowadays, there is a constant high demand from services, which makes scaling a great deal for maintaining good performances. However, scaling VMs is not practical and not so easy since they are heavy and take a long time to boot, as they contain complete and isolated operative systems.

Meanwhile, *Microservices* have become more and more popular. It is an architectural design that breaks an application into independent, loosely-coupled, and individually deployable services [5], where each service holds a specialized set of related functions. Also, another level of virtualization, *containers*, has been created and become a huge trend for usage with microservices architecture [6].

Containers are packages of software that include all that is needed to run an application including code, dependencies, libraries, and more [7]. To run them there is a popular software, Docker, which allows the creation and execution of container images. Containers are similar to VMs, but more lightweight, portable, and much faster to boot [8] which enable better scalability. Thus, the characteristics of containers and the principles of microservices led to a partnership that is highly seen nowadays, where microservices applications are composed of multiple units, deployed in containers. However, the division of functions amongst several components may generate a high number of smaller ones which increases complexity.

As complexity grows, a need to run and manage several containers emerges [9]. This is where container orchestrators, like Kubernetes [10] and Docker Swarm [11], come into the scene. They are responsible for the shutdown, addition, and assignment of containers to cluster nodes. A cluster is a set of machines, known as nodes, that might be spread across the Internet. Right now, cluster and container management are being offered by cloud providers, like Amazon and Microsoft, through Kubernetes (Amazon Elastic Kubernetes Service (EKS) [12] and Azure Kubernetes Service (AKS)[13] respectively).

Microservice-based and other types of applications must have their behavior monitored so that it becomes possible to anticipate or detect production issues, make decisions to face identified threats and improve performance [14]. **Self-adaptation** is a risk mitigation strategy for uncertainties induced by runtime changes [15]. Thus, if applications possess self-adaptation mechanisms, they will be capable of gathering information from the environment and themselves, reasoning on that information, and adapting to meet goals. All this without intervention from a supervisor. Thereby, management activity costs are reduced once supervisors do not have to be constantly monitoring resources and performing manual adaptations. Currently, one of the most famous open-source monitoring tools is Prometheus [16] which can be integrated with a powerful tool like Grafana [17] for visualization of data.

The obstacles with current monitoring and actuating solutions are limited applicability and the absence of a complete management loop responsible for monitoring and adapting systems. This is where TMA [18] becomes useful. TMA is a cloud monitoring and management platform flexible on the metrics it collects and on the adaptations it triggers. It works and dispatches adaptations based on quality models which are weighted metrics trees. TMA allows every system to be monitored and adapted in every way we want as long as support is given for that from probes and actuators. Probes are agents that collect metrics from monitored systems, while actuators are the agents responsible for executing the adaptations on the managed system. There is a User Interface (UI) developed for TMA, however, it had the basic purpose of creating noncomplex configuration data. As an example, the creation of the metrics and quality models, which are more complex operations within the scope of creating entities, are not present in that interface. Besides, it seems to be quickly developed without sufficient care about its aspect. Still, the biggest and main issue of it is the absence of visual features that support and facilitate the systems management activity.

TMA is being used in the context of the **TalkConnect** project, which is a project that aims at providing a telecommunications system with the ability to add and remove resources according to demand. Thus, a dashboard would ease the management of the telecommunications system.

At the moment, decision support features on the current TMA's interface is nonexistent. And functionalities in that sense would be a great help when managing systems. Thus, the main goal of this dissertation is to plan, design, and develop a new UI for TMA that provides visual support for the decision process when managing systems.

1.1 Objectives and Approach overview

The goal of this dissertation is to design and implement a UI, as a dashboard, which will be used as a decision support tool for TMA's administrators when monitoring and managing cloud systems. Also, this interface should allow the creation of complex TMA's entities (such as metrics and quality models), and manage adaptation rules. With that being said, in summary, the objectives for this work are:

- Create a user interface, in the form of a dashboard, that will allow create metrics and quality models, create and manage charts of metrics, simulate metrics values, and manage adaptation rules;
- Extend TMA's API component;
- Reprogram TMA's Analyze component to automate the process of analyzing systems metrics with different quality models;
- Change and implement TMA's Planning component architecture.

TMA is a cloud monitoring and managing platform researched and developed in the group Software and Systems Engineering (SSE) of University of Coimbra (UC) that enables self-adaptation on systems, cloud-hosted or not. The integration between the interface and TMA happens through the existing TMA-API component which was complemented to support new features. This means all interface operations that require communication with TMA are turned into API requests served by the TMA-API component.

The interface is a web page in the form of a SPA, which is a web application implementation that only loads a single web document, and then updates the interface via JavaScript. [19]. It should be available on the network, providing TMA administrators with the following systems managing features:

- Create and visualize TMA's database entities such as metrics, quality models and configuration profiles;

- Plot collected metrics from monitored systems and optionally the plans alongside;
- Simulate metrics values;
- Export charts as images;
- Export charts configurations;
- Manage homepage charts;
- Manage adaptation rules.

As mentioned before, new endpoints were added to TMA's API component. Also, TMA's Analyze component was automated to read the quality models from the database instead of having them hard-coded and the processes to apply on the metrics data were modulated. Respecting TMA's Planning component, its architecture was changed. An API server was added to receive requests concerning adaptation rules and those rules can now be updated on the fly and they are read and saved on the database instead of a file.

After the implementation of the dashboard and the other TMA components, validation tests were performed on usability, performance, and functionality. Functionality and usability validation was applied on the dashboard, while performance was applied on the dashboard and the TMA's API component.

1.2 Document Structure

The rest of the document is organized as follows:

- **Chapter 2**: This chapter presents concepts related with this dissertation and the tool to be developed. First, a set of monitoring tools (generic, infrastructure, and container) are described. Following, concepts related to trending technologies used within the Cloud like microservices and containers are provided. Finally, as a choice of frameworks for this dissertation was needed, several frontend development frameworks are described, and their benefits and drawbacks are presented. And at the end a comparison between those frameworks is made to reason on a choice;
- **Chapter 3**: This chapter presents the concept of self-adaptive systems and a set of existing self-adaptive enabling tools. Then, a comparison between them is made to choose one for this work. Additionally, a work performed within this dissertation that is related to self-adaptive systems is summarized and at the end, an observation is made on why to use the chosen self-adaptive enabling tool.
- **Chapter 4**: This chapter summarizes the functional requirements elicited for the product with the support of use case diagrams. Also, an implementation status of those requirements is presented. Following details concerning

non-functional requirements and design and implementation restrictions are provided;

- **Chapter 5**: This chapter initially presents the architecture proposed for the tool. Then, details about implementation are provided, starting with maintenance and ending with the implementation of requirements;
- **Chapter 6**: This chapter initially describes how a brief validation of TMA was performed. Then, it explains how the functional and non-functional requirements were validated along with their results and discussions;
- **Chapter 7**: This chapter, in the first place, presents and describes the development model followed. Then, details on management-related activities performed such as requirements prioritization and management, planning of tasks, and risk analysis are presented;
- **Chapter 8**: This chapter summarizes and highlights the main ideas and results from this dissertation, and presents future work to be made on the product developed.

Chapter 2

Background

Nowadays, the *Cloud* is seen as a resource that promotes fairness and cost savings through its pay-as-you-go business model [1]. Consequently, instead of supporting the costs of building and maintaining their infrastructure and data centers, companies started renting access to cloud infrastructure to provide their services worldwide [20]. Despite all the main advantages the cloud provides, such as scaling, upfront cost savings, and resource overprovisioning reduction, it also raises concerns for companies when it comes to security and performance since applications and systems are dependent on the cloud provider. Besides, the *Cloud* is an environment that may be shared by different users and, thus, monitoring becomes essential for a client to trust the environment and to manage the performance of his applications.

In this chapter, concepts the tool to develop is involved with, and that are related to this internship will be presented. First, a set of monitoring tools are described. Then, concepts related to trending technologies quite used within the Cloud like microservices and containers are provided. Finally, several frontend development frameworks are described along with their benefits and drawbacks. And a comparison between them is made to choose the ones to use in the development of the new interface for Trustworthiness Monitoring & Assessment Framework (TMA).

2.1 Monitoring Tools

Cloud monitoring is seen as the process of evaluating the state of cloud-provided infrastructure. Using automated and manual tools, it is possible to manage, monitor, and evaluate cloud computing architecture, infrastructure, and services. With these tools, it becomes possible to track the performance, safety, and availability of crucial cloud apps and services. They allow administrators to monitor the status of cloud-based resources, helping to identify emerging defects and troubling patterns so minor issues can be prevented from turning into significant problems [21].

There are different options when it comes to cloud monitoring tools which can be categorized by the type of cloud services they monitor. However, even being part of the same category of monitoring tools, they have different characteristics.

The next sections present some of the most popular open-source tools for monitoring cloud-based resources, grouped in categories. Those categories are *Generic Monitoring Tools*, *Infrastructure Monitoring Tools* and *Container Monitoring Tools*.

2.1.1 Generic Monitoring Tools

In this section, tools which are not restricted respecting monitoring are presented. Meaning, that these tools can be used to monitor any type of system and/or infrastructure.

Prometheus

Prometheus is an open-source monitoring solution focused on gathering and analyzing time-series data [22], meaning that data collected from systems are stored along with their timestamp of recording.

The fundamental data unit used is a metric to which a name, so it can be referenced, and a set of labels are assigned. Labels are arbitrary key-value data pairs that can be used to filter the metrics in the database [23].

Prometheus presents the following features [16]:

- A multi-dimensional data model with time series data identified by metric name and key/value pairs;
- A flexible query language, PromQL, to leverage the multi-dimensionality of the feature above;
- No reliance on distributed storage; single server nodes are autonomous;
- Data collection is based on a pull model over HTTP;
- Support for pushing data through an intermediary;
- Monitoring targets can be discovered either by service discovery or static configuration;
- Support for multiple modes of graphing and dashboarding.

Prometheus architecture is depicted by Fig. 2.1. Its ecosystem is composed of multiple components, but some are optional. Following a brief description of those components is made:

- **Prometheus server:** Responsible for pulling, processing, and storing data collected from systems;

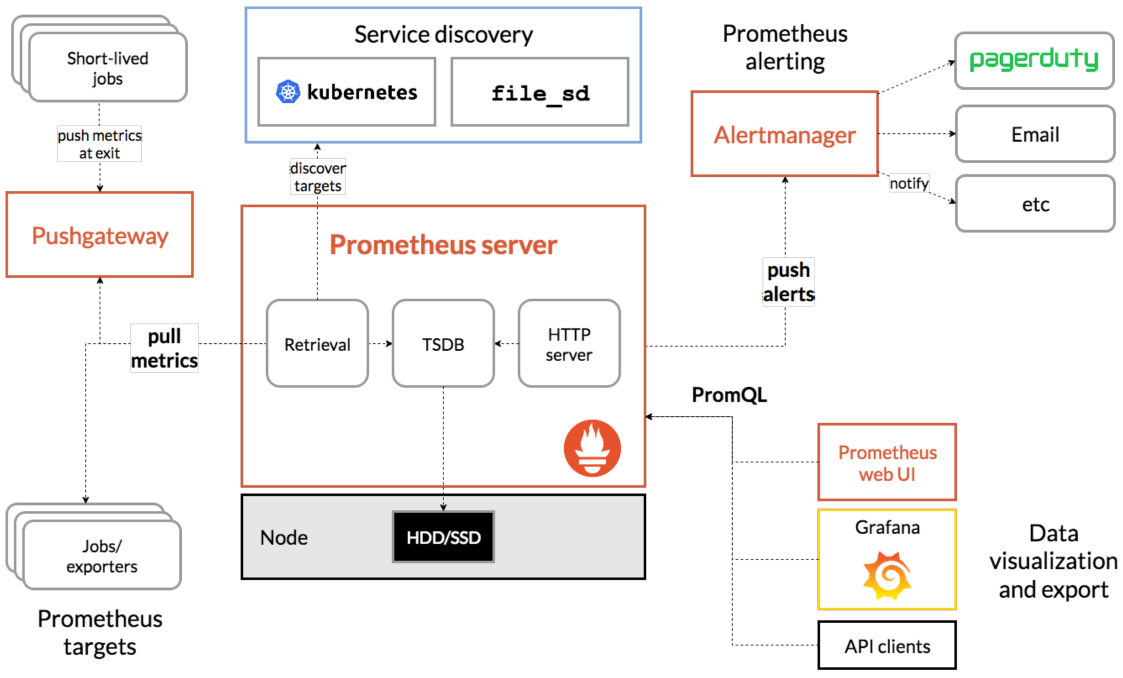


Figure 2.1: Prometheus architecture (from [16]).

- **Client libraries:** Code meant to be integrated into the code of an application to allow exposing, by an Hypertext Transfer Protocol (HTTP) endpoint, the metrics collected;

- **Push Gateway:** This component gives support for exposing short-lived jobs, since they may finish before the Prometheus server pulls the metrics data. Thus, ephemeral jobs can expose their metrics by sending them to this gateway;

- **Exporters:** They work similar to client libraries, by allowing to expose metrics. However, these are built especially for cases where is not possible or feasible to collect metrics directly as with *client libraries*. Basically, they are integrations that allow converting existing metrics from third-party systems into Prometheus metrics;

- **Service discovery:** Responsible for providing *Prometheus server* with the monitoring targets and their endpoints for pulling metrics;

- **Alert manager:** This component allows sending notifications. According to rule-based policies defined in the *Prometheus server*, alerts may be sent to this component. Then, it charges itself for delivering the notifications through messaging and mailing services. Besides, these components provide support for aggregating and muting repetitive alerts, thereby preventing spam of notifications when multiple events occur in a short time frame;

- **PromQL:** This is a query language that allows retrieving stored data from *Prometheus server* and integrating it with dashboards for building charts and performing analysis.

Graphite

Graphite is a popular open-source monitoring tool compatible with any type of system, whether cloud-based or not [24]. It is used to monitor cloud applications, platforms, and infrastructure.

What Graphite does not do is perform the process of collecting data or store it persistently, however, it can be integrated with several other tools that do [25]. Thus, Graphite is mainly focused on the operation of storing the data being collected and on rendering graphs of those data as they are received.

Its architecture, presented by Fig. 2.2, is pretty simple and easily explained:

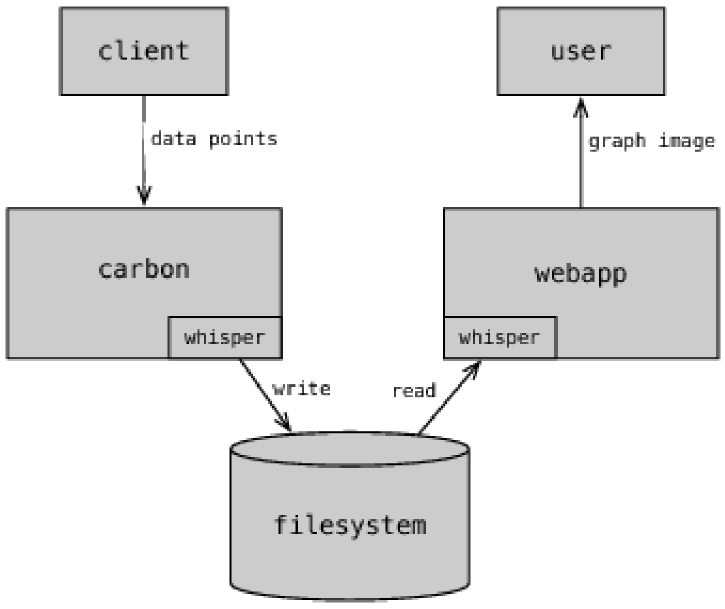


Figure 2.2: Graphite architecture (from [26]).

- **Carbon:** This is a daemon responsible for listening to incoming time-series data from clients. Upon receiving, those data is persistently saved on disk using the Whisper component;
- **Whisper:** This element is simply a database driver that will allow communication with a database either to write, update or read data;
- **Graphite Webapp:** This component provides a User Interface (UI) which renders graphs on-demand. As soon as data is received by the *Carbon* component, very quickly it becomes available for graphing. Then, this web app offers several ways to create and display graphs including a simple URL API for rendering that makes it easy to embed graphs in other webpages [27].

Although Graphite already provides a UI, it is not that much of a great user experience provider. However, it is possible to integrate it with more elegant graphing tools, like Grafana [28].

2.1.2 Infrastructure Monitoring Tools

In this section, the tools presented are restricted in their applicability. This means that these tools can only be used to monitor system resources such as Central Processing Unit (CPU), Random Access Memory (RAM) and disk.

Zabbix

Zabbix is an open-source monitoring tool used to monitor servers, networks, Information Technology (IT) components, cloud services, and virtual machines. It provides monitoring metrics such as the network utilization, consumption of disk space, and CPU load [29].

Zabbix uses a flexible notification mechanism that allows users to configure e-mail-based alerts for virtually any event, and it offers excellent reporting and data visualization features based on the stored data [30].

All Zabbix reports, statistics, and configuration parameters are accessible through a web-based frontend. This frontend allows checking the status of the IT infrastructure from anywhere using the Internet.

Zabbix's architecture can be one of two types, centralized or distributed. Fig. 2.3 presents the distributed version, where the difference is that the Zabbix agents instead of directly communicating with the Zabbix server, communicate with the Zabbix Proxy. Having this consideration, a Zabbix architecture is composed of the following elements [31]:

- **Server:** This is the central component to which *Agents* report metrics, in case of a centralized approach. In the case of following a distributed approach, *Proxies* are the ones that send the metrics. This component is the one that provides access to the central repository in which all configuration, statistical and operational data are stored;
- **Database Storage:** This is the component that holds all configuration information and the data collected by Zabbix agents;
- **Web interface:** Component that allows accessing Zabbix from anywhere through a web-based interface;
- **Proxy** This component plays the role of intermediary, collecting metrics from *Agents* on behalf of the Zabbix server. A proxy is an optional part of Zabbix deployment, implemented when following a distributed approach to take advantage of distributing the load from the **Server** to the *Proxies*;
- **Agent:** These components are the ones that actively collect metrics from targets being monitored. Then, these components report those metrics to the *Server* component.

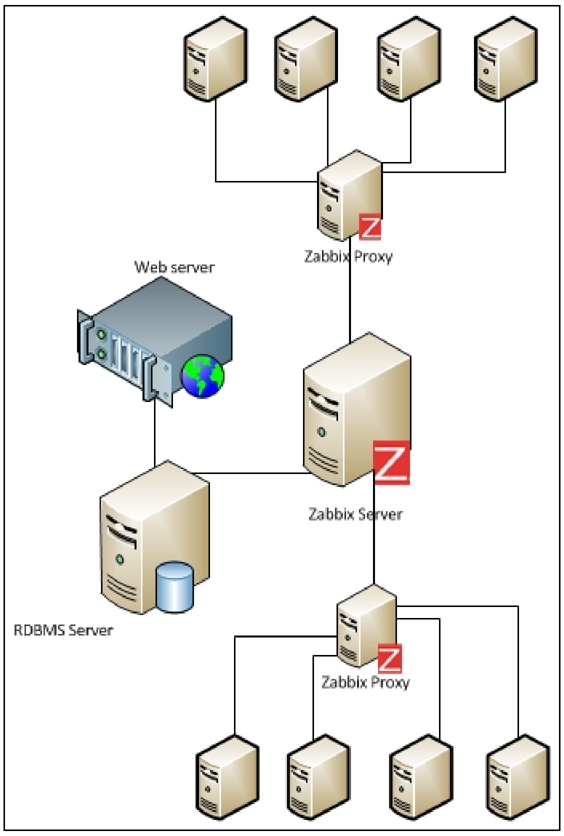


Figure 2.3: Zabbix architecture (from [32]).

Nagios

Nagios is an open-source tool for monitoring systems, networks, and infrastructures [33]. It periodically invokes plugins hosted on monitored resources to collect and retrieve information about the current state [34].

Upon retrieve of information from plugins, Nagios processes that data against defined rules and, if a pattern is detected, alerts about the issues found so that the technical team may perform healing processes.

Nagios follows a server-agent architecture, presented by Fig. 2.4, composed of the following elements:

- **Nagios server:** Component installed on a host, responsible for executing a scheduler that periodically asks for monitoring data from plugins;
- **Plugins:** Elements which are incorporated into the resources to monitor and responsible for collecting and sending the data to the server when asked of them;
- **Web interface:** Component that provides a graphical visualization of the state of the resources being monitored, and constantly updated according to data sent by plugins.

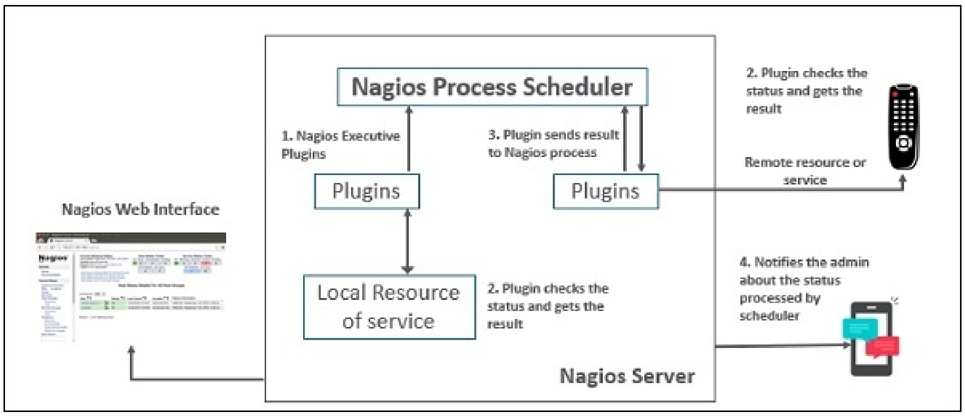

Nagios Process Scheduler
1. Nagios Executive Plugins
3. Plugin sends result to Nagios process
2. Plugin checks the status and gets the result
Remote resource or service
Plugins
Plugins
Nagios Web Interface
Local Resource of service
2. Plugin checks the status and gets the result
4. Notifies the admin about the status processed by scheduler
Nagios Server

Figure 2.4: Nagios architecture (from [34]).

2.1.3 Container Monitoring Tools

In this section, tools that can only be used to monitor containers are presented.

cAdvisor

cAdvisor is an open-source tool that provides resource usage, performance characteristics and related information about the containers running on the cloud [35].

By running a background process that collects and processes information from containers, this tool becomes very helpful in providing insights about resource consumption from the code running on the containers. Thus, it aids in identifying performance bottlenecks and performing changes to enhance the scalability of systems.

This tool's architecture is very simple. An instance of cAdvisor runs inside a host and then it collects metrics from all the containers co-located within that host. The data collected by a cAdvisor instance can be accessed by a web-based UI. Fig. 2.5 presents a cAdvisor architecture as just described along with integration with Prometheus.

At the web UI provided by cAdvisor, the container information that can be seen are e.g., the container names, their resource usage statistics within the last period, and the histogram of resource usage. Besides container information, cAdvisor also collects data respective to the machine which can also be consulted at the web UI, such as the number of logical core CPUs, memory capacity, and network devices.

2.1.4 Summary

As seen in Sec. 2.1, monitoring is a very important activity when maintaining applications or infrastructure. As monitoring consists of having insights into the system's current state, it becomes a useful activity to perform so that adjustments

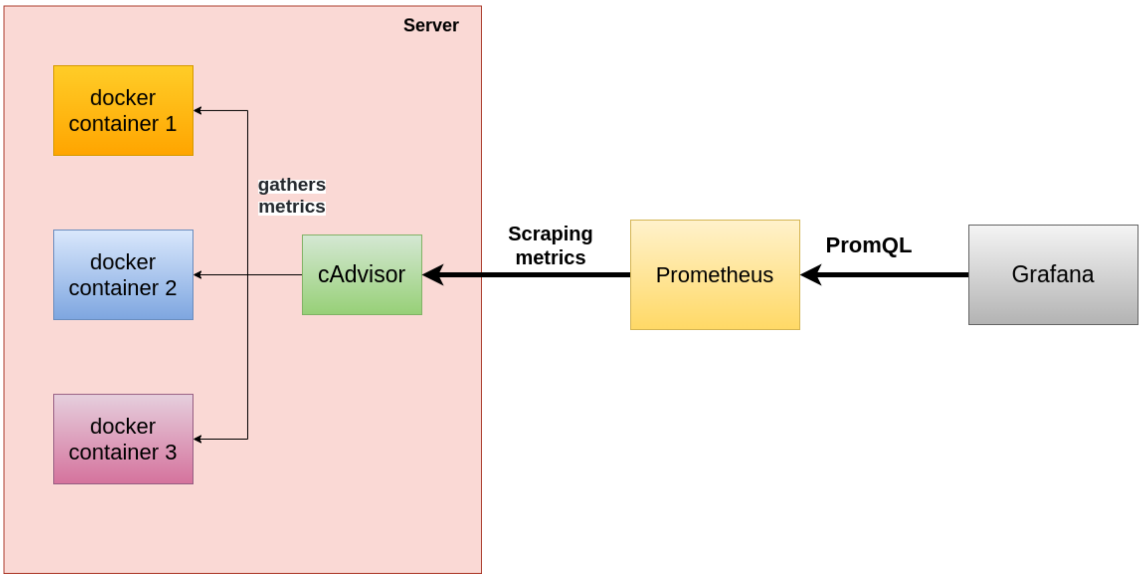


Figure 2.5: cAdvisor architecture, integrated with Prometheus and Grafana (from [36]).

can be made to deal with threats that may be identified.

However, in that same section, it was also seen that adaptations are not automatically performed. They always have to be triggered manually which requires allocating more time from IT professionals to the systems management activity. Nonetheless, some of the tools described may become very handy and helpful when a system, or infrastructure, needs to be analyzed in depth.

In any case, as these tools do not provide complete support for the whole management activity life-cycle, by not including adaptation features, self-adaptive systems become a need to improve the management task. By being able to apply self-adaptive capabilities to systems, human resources wasted in the management activity can be decreased and allocated elsewhere, to other activities also important.

2.2 Microservices

Microservices is an architectural style that structures an application as a collection of services [37]. This type of architecture is meant for building distributed applications as an application is split into independent, loosely-coupled, and individually deployable smaller services. This allows each service to be scaled or updated without disrupting other services that compose the application. Consequently, it enables fast, frequent, and reliable delivery of large and complex applications [38].

2.2.1 Monolithic vs Microservices Architecture

The monolithic architecture is considered to be a traditional way of building single and indivisible unit applications [39]. Its set of functionalities is encapsulated into one single application and, thereby, its modules cannot be executed independently, making everything tightly coupled and all logic for handling requests present in a single process [40].

Usage of basic features from a programming language to divide and organize the application into classes, functions, and namespaces may be used, but developers feel frustrated with monolithic applications, even more as they are being deployed to the cloud. This frustration begins with tied change cycles, once a change is made to a small part of the application requires the entire monolith to be rebuilt and deployed. Over time it becomes hard to keep a good modular structure and even harder to keep changes that only affect one module, within that specific module. Besides, scaling is applied to the whole application rather than to parts of it, making the resource requirements greater than what they should be [41].

Although having its strengths of being a single unit, such as **less cross-cutting concerns (e.g, logging and caching)**, **easier debugging and testing** and **simpler deployment and development**, monolithic applications have some drawbacks which motivate the usage of microservices:

- **Understanding -** Applications may become difficult to understand and modify and, therefore, development typically slows down;
- **Continuous development -** Difficult continuous development once a change made on a small part of the code may require changes on the whole application;
- **Scalability -** Scaling produces unnecessary costs since only a part of the application is critical and requires extra resources, but the whole has to be scaled;
- **New technology barriers -** Requires a long-term commitment to a technology stack, making it extremely problematic to apply a new technology since the whole application has to be rewritten.

Contrarily to these drawbacks, *Microservices* have the advantages of being **independent components** and having a **better scalability and understanding**, since each component has its subset of related functions from the whole application. Also, **resilience and lifecycle automation** are other strong and motivating characteristics, once the failure of a component and its development and deployment do not mess with the functioning of the other components. But like everything, *Microservices* has its downsides:

- **Complexity -** Microservices architecture is a distributed system. That means multiple connections have to be chosen and set up between all components and databases which leads to additional care in handling communication and preventing errors from disrupting other services. Consequently, testing complexity increases as more test cases are needed in each component;

- **Solving errors -** As there are many communications between multiple units, finding and tracing errors can consume a lot of time by just looking for those errors;
- **Cross-cutting concerns -** Since application functions are spread across multiple components, cross-cutting concerns (e.g., external configurations, logging, and health checks) must be carefully treated and thought;
- **Testing -** Once there are multiple units prone to failures, the testing activity becomes much harder on a microservices application.

Because of all the benefits and downsides of both architectures, it is suggested to start a project using a monolithic architecture, even though it is foreseen that microservices should be implemented. The reason for this is that starting development with a monolith allows exploring both the complexity of a system and its component boundaries. Then, if the complexity begins to grow and becomes hard to handle, it may make sense to convert it into microservices [42].

2.3 Containers

Containers are a solution to the problem of how to get the software to run reliably when moved from one computing environment to another [43].

A container can be seen as one package that consists of an entire runtime environment. It accommodates an application and any kind of dependencies and configuration files needed to run. Thus, a containerized application allows to abstract from differences in operative systems and underlying infrastructure.

Both Virtual Machines (VMs) and containers represent types of virtualization. However, while VMs can have a size of some gigabytes, a container is much lightweight generally possessing tens of megabytes. As a consequence, a single server supports hosting far more containers than VMs.

Another benefit of using containerized applications is that they can boot up almost immediately, while VMs may take several minutes to start their operating systems and begin running the applications they host. Once containers can then be instantiated very quickly, resources can be spared by removing containers when they are not needed anymore and re-add them when they are.

Finally, containerization promotes modularity meaning that instead of running an entire complex application, only a single module is run. The idea is to follow the microservices architecture, where an application is split into modules, each responsible for a subset of related functions from the application as a whole. By doing this, applications become easier to manage because each module is relatively simple, and changes can be applied to modules without having to rebuild the entire application. And as containers are so lightweight, instead of instantiating the whole application, it is possible to immediately instantiate parts from the whole application.

The next sections will present and dive into more detail about some of the most popular container technologies.

2.3.1 Docker

Docker is an open-source platform for developing, shipping, and running applications [44]. The platform it provides, along with multiple tools, serves the purpose of managing the lifecycle of containers.

Docker enables separating the infrastructure from the application code so that software is quickly delivered. With Docker, infrastructure and applications can be managed similarly. Leveraging Docker's methodologies for shipping, testing, and deploying code quickly, the delay between writing code and running it in production is reduced.

Docker uses containers and thereby it enables packaging and running applications in loosely isolated environments. The isolation provided allows running many containers simultaneously on a given host.

Docker follows a client-server architecture, which is represented by Fig. 2.6, composed of the *Docker Client*, *Docker Host*, *Docker Objects*, and *Docker Registry* components [45]:

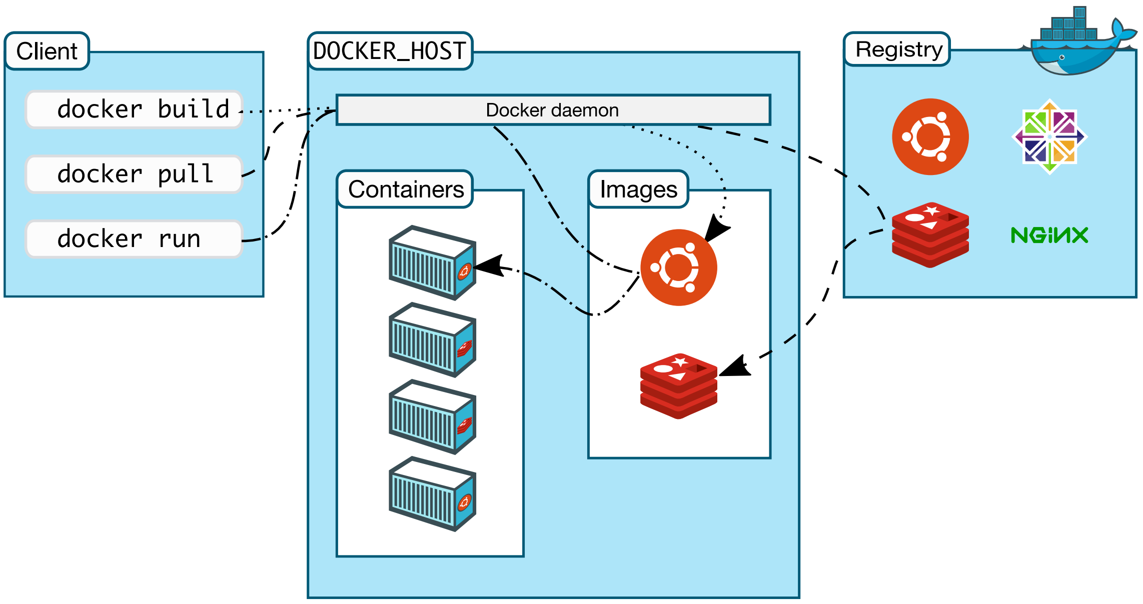


Figure 2.6: Docker architecture (from [44]).

- **Docker Client:** This component is responsible for allowing users to interact with Docker. The Docker client may be present on the same host as the Docker daemon or be connected to a daemon on a remote host. Moreover, a docker client can communicate with multiple daemons.

The Docker client component provides users with a Command Line Interface (CLI) that enables sending commands to the daemon, issuing build, run, and stoppage of applications. Basically, it provides mechanisms for a user to direct the pull of images from a registry and have them running on Docker hosts;

- **Docker Host:** This component provides an environment to execute and run applications. It is composed of a Docker daemon, and Docker objects such as images, containers, networks, and volumes. The daemon element is responsible for listening to requests and managing Docker objects. The daemon can also communicate with other daemons to manage its services.

  The Docker daemon pulls and builds container images as requested by the client. When a requested image is pulled, it builds a container following a set of instructions defined in a build file. This file may include instructions for loading other components before running the container, or instructions to be sent to the container's local command line once it is built;

- **Docker Objects:** Different types of objects are used when assembling the containerized applications. The main Docker objects are: *Images*, *Containers*, *Networking* and *Volumes*.

  Images are templates that can be used to build and run containers or serve as the base for inserting additional behavior to create other images. Moreover, images are built according to instructions defined by Dockerfiles which are files containing the steps to build images and insert their dependencies.

  Containers are the encapsulated environments in which images are instantiated and applications run. Besides the configurations defined in the images, when containers are built and run, additional configurations may be provided.

  Networking allows defining different types of networks in which containers can recognize each other and can talk to.

  Volumes are some kind of extensions to the containers' default behavior, that allows containers to persist their data even after they terminate;

- **Docker Registry:** is a service that provides locations where images can be stored and downloaded. That is, it is a repository that hosts and provides access to Docker Images. Docker provides one of these repositories called Docker Hub.

2.3.2 Kubernetes

Kubernetes is an open-source container orchestration platform that automates many of the manual processes involved in deploying, managing, and scaling containerized applications [46].

In Kubernetes, the smallest deployable units of computing that can be created and managed are *pods* [47] and they are composed of one container or a group of them. Also, *pods* are created from specifications that are defined in template files, written in YAML Ain't Markup Language (YAML).

Essentially, Kubernetes is composed of a cluster that is formed by a set of worker machines, known as nodes, and a control plane component. The control plane is responsible for making global decisions about the cluster, managing worker nodes, and *pods*. The control plane component can be run on any machine in the cluster, however, to simplify things, the standard setup hosts all of its components on the same machine, and does not allow that machine to host any other containers that might be created by the user. The worker nodes host and run containerized applications, more precisely *pods* which are components that constitute an application.

Diving into more detail on kubernetes architecture, which is presented by Fig. 2.7, there are 2 major components, as just presented, which are the control plane and the worker nodes. However, for each of those components there are several smaller elements that constitute them. Next, those smaller elements are explained:

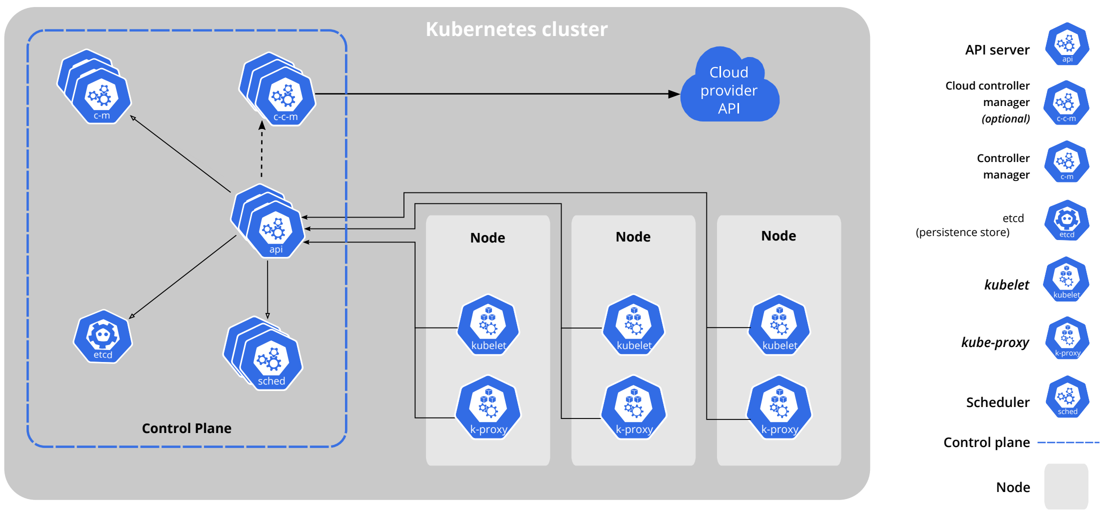


Figure 2.7: Kubernetes architecture (from [48]).

- **kube-apiserver:** This element is part of the control plane and its purpose is to expose the Kubernetes API. It essentially is the front end for the Kubernetes control plane;
- **etcd:** This element is part of the control plane and is composed of key-value storage that is used by Kubernetes to save all cluster data;
- **kube-scheduler:** This element is part of the control plane and it is responsible for allocating pods on worker nodes, based on additional information it may have, e.g., resource constraints;
- **kube-controller-manager:** This element is part of the control plane and it is in charge of running controller processes, such as detecting and responding to node failures, ensuring completion of jobs, exposing pods, and controlling access to services;

- **cloud-controller-manager:** This element is part of the control plane and its function is somehow similar to the *kube-controller-manager*, but for cloud-specific controller processes. It allows associating the cluster to a cloud provider's API, distinguishing components that interact with that cloud platform from the ones that do not. Thus, it implements the logic that is specific to the cloud provider on the components that are linked to it;

- **kubelet:** This element is part of the worker nodes and it is a process that ensures containers are running in a Pod. It receives information from all the pods running on the host and, based on that, it checks that containers are running and healthy;

- **kube-proxy:** This element is part of the worker nodes and it is a network proxy. It maintains network rules that define how communication can, and cannot, happen with the pods from inside or outside connections of your cluster;

- **Container runtime:** This element is part of the worker nodes and it is simply the software used for running containers, which can be e.g., Docker.

2.4 Frontend Development Frameworks

In this internship, the main goal is to build a UI that will aid in the creation of TMA's complex entities (metrics and quality models) and provide decision support when managing systems to define adaptation rules and plans. The idea is to have visual features such as the view of metrics trees, charts with metrics values, and even simulations of the same quality model but with different weights.

However, there is a lack of experience in designing and implementing web pages and user interfaces. So, it becomes necessary to carefully choose a framework that can facilitate my development. For that, the general support provided for developing over that framework must be considered, as well as the existence of libraries for developing the most complex features, such as the representation of weighted trees of metrics.

Frameworks have become an essential part of web development. As the standards of web applications are always rising, so does the complexity of the technology needed [49]. It is not a good approach to reinvent sophisticated and complex techniques when there are frameworks endorsed by thousands of developers around the world. Thus, leveraging frameworks is a path that leads to richer and more interactive web applications.

A Web application has a backend (server-side) and a frontend (client-side). In this dissertation context, there is already a backend, the TMA's API component. Thereby, it makes sense to stay on this track and leverage what has already been built. Although being necessary to add functionalities and adjust this component to support what the UI will allow, a frontend framework choice must be made. Thus, the next sections will present some of the most popular front-end frameworks, along with a description, and at the end, a comparison between them will be made to make a decision.

2.4.1 Angular

Angular was created and is being maintained by Google. It is an open-source, JavaScript framework written in TypeScript, that provides a standard structure for developers to create large applications in a maintainable manner [50]. It is based on the Model-View-ViewModel (MVVM) pattern, which is composed of three components [51]: *View* which defines how data is displayed graphically, *Model* which is responsible for accessing different data sources, and *ViewModel* which implements the *View* logic in response to user interactions and coordinates the interaction between the other two components by passing all the necessary data. Using MVVM pattern, Angular ensures two-way data binding for immediate synchronization between the model and the view components, so any change in the view will instantly reflect in the model and vice-versa.

Angular also provides a feature, called *directives* which allows developers to program special behaviors for the Document Object Model (DOM), thus making it possible to create rich and dynamic HTML content. There is, too, a hierarchical dependency injection feature, which makes code components highly testable, reusable, and easier to control. This feature also helps define code dependencies as external elements, decoupling components from their dependencies.

Angular applications architecture, depicted by Fig. 2.8, is composed of 8 main building blocks:

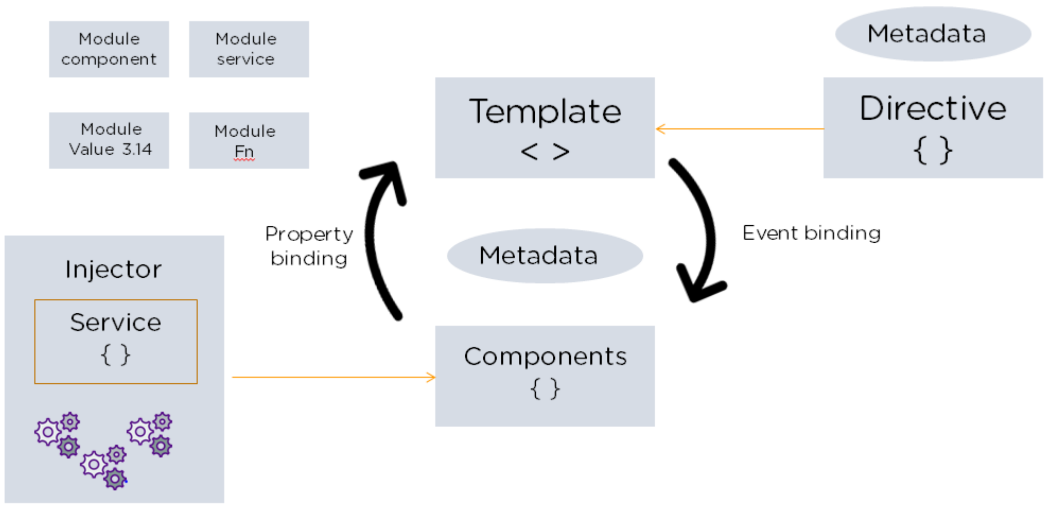


Figure 2.8: Angular architecture (from [50]).

- **Modules:** Every Angular application has a root module, conventionally named AppModule, which provides the bootstrap mechanism that launches the application. An app typically contains many functional modules and in case we want to use another custom Angular module, then it needs to be registered inside the app.module.ts file. Organizing code into distinct functional modules aids in managing the development of complex applications, and in designing for re-usability [52];

- **Components:** Components define classes holding application logic and data. Usually, a component is part of the UI;
- **Templates:** This architecture component combines Angular markup with HTML to modify HTML elements before they are displayed. Template directives provide program logic while binding markup connects application data to the DOM. There are two types of binding: **i) Event binding -** where a target is updated with response to user input; **ii) Property binding -** Allows assigning values to HTML elements properties;
- **Data binding:** Data binding allows communication between a template and its component. It is also important for communication between parent and child components. Angular allows communications between components and the DOM, making it very easy to create interactive applications without worrying about pulling and pushing the data;
- **Directives:** They can be seen as markers on the DOM element, giving instructions to Angular for attaching a certain behavior to the element, or even changing it. Directives are classes that add additional behavior to elements. For example, Angular's built-in directives can be used to manage forms, lists, styles, and what users see;
- **Metadata:** This is used by Angular to understand how to process a class. Metadata are decorations applied to classes from which Angular can configure their expected behavior;
- **Services:** When you have data or logic that isn't associated with the view but has to be shared across components, a service class is created. The class is always associated with the @Injectible decorator to provide metadata that allows the service to be injected into components as a dependency. That is how components are distinguished from services to increase modularity and reusability;
- **Dependency Injection:** It allows maintaining component classes lean and efficient. It is an architecture block responsible for injecting services. Thus, components can access the services' set of functionalities.

In the end Angular applications carry some advantages such as:

- **Component-based architecture**: This allows re-usage and easier maintainability;
- **Two-way data binding**: This binding ensures any changes in the DOM (view) get reflected in the application data and vice versa. This leads to less code writing since developers do not have to worry about manually updating the view or the application data;

- **Dependency injection**: As services just need to be injected where they are needed, the writing of modular services is allowed, decoupling them from components;
- **Testing**: Since Angular has been built from the start with testability in mind, testing activities are eased. Tests are first-class tools, and thereby it is possible to test every part of the application;
- **Development support**: Since there is a huge community and support materials, there is a lot of support when developing.

On the other hand, it also has the following downsides:

- **Steep Learning Curve**: Angular is verbose and complex which makes it hard to learn. Besides, it is a total dynamic solution with multiple ways of doing the same operation;
- **Bloated code and large size**: The complex structure and size of Angular applications sometimes is the reason why dynamic apps do not perform well;
- **Limited Search Engine Optimization (SEO) capabilities**: In terms of search engine crawlers, Angular provides poor accessibility.

Summing up, Angular seems to be good for creating large-scale, loosely coupled, and well-structured applications. However, if the application to build is simple, Angular may be too overwhelming and complex to learn, thus becoming a better option to use an easier framework.

2.4.2 React

React is considered to be one of the most popular front-end frameworks. In fact, it is a library, because to be considered as a framework it would need to have a core package of functionalities usually needed when building a web app and it does not. Generally, when creating react applications, other React-specific libraries have to be included [53].

React is an open-source JavaScript component-based library featuring JavaScript Syntax Extension (JSX) syntax. It was developed by Facebook to deal with maintainability problems caused by the app's continual inclusion of new features [54]. The core feature of React is its virtual DOM with one-way data binding. Due to the virtual DOM functionality, React is praised for its superior performance. It is being continuously updated and, at the moment, programming in React no longer requires classes, instead, hooks can be used to produce cleaner and faster coding. It is considered to be one of the easiest frameworks to learn thanks to its user-friendliness and comfortable learning curve.

As pointed out before, React is a library and therefore it does not maintain some important features. That's why it must work together with other libraries, such as for state management, routing, and interaction with APIs.

Although there is a set of suggested practices, React does not have a specific architecture. However its behavior can be explained, as illustrated by Fig. 2.9, by the following set of components and functions:

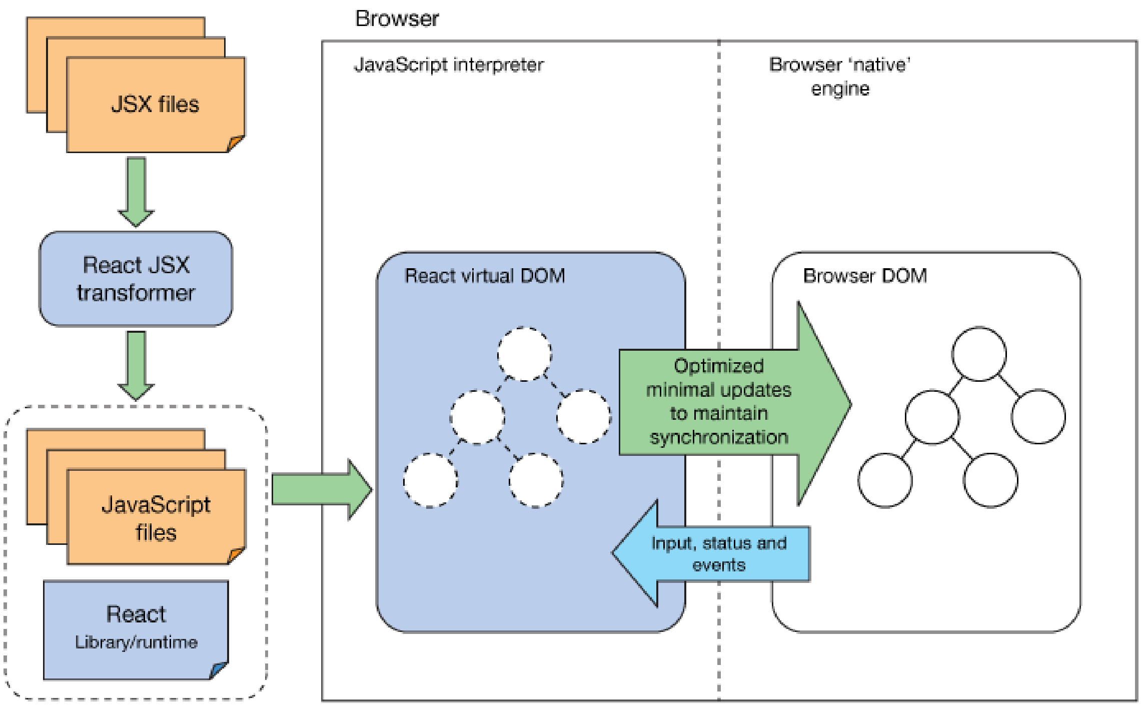


Figure 2.9: React's behaviour (from [55]).

- **JSX files:** JSX is a syntax used by React which extends JavaScript to include HTML text so that both can co-exist [56]. This allows writing HTML structures in the same file JavaScript is written, connecting UI design with data handling. JSX files are used to design components and their behaviour;
- **React JSX transformer:** Browsers do not understand JSX, thereby it becomes essential to have a compiler that converts JSX to regular JavaScript [57];
- **Virtual DOM:** The virtual DOM like the Browser's DOM holds a tree of objects that represent a view. When a change is made to the tree, or to a node, that change is firstly processed by the React's virtual DOM. Then, this virtual DOM performs an optimized diff of its internal state against the browsers' DOM and performs the minimal updates required to keep the UI consistent [55];
- **Browser DOM:** It is the actual tree of objects a user will interact with. When inputs or events are provided, they are passed to React's DOM to be handled.

We can take away the following advantages of using React as a support for developing frontends:

- **High-speed operations in the DOM** allowed by the usage of a Virtual DOM;

- **Compatibility with other JavaScript libraries** to extend and improve the interaction of the user with the view since React itself is a library;
- **Easy to learn and use** as there are a lot of tutorials and training materials, which make it a good choice for beginners or less experienced developers;
- **Testability, maintainability, and reusability** since React applications are component-based, thus decoupling functions into well-defined scopes;
- **SEO Friendly** once the view is rendered in the virtual DOM, but returned to the browser as a regular web page with the help of *React JSX transformer*.

Despite having some appealing advantages, it has the following drawbacks:

- **Frontend only coverage** as React only deals with and supports the development of the view and not the backend of an application;
- **JSX as a barrier** for new developers or inexperienced ones with JavaScript;
- **High pace of development** which leads to poor and unfinished documentation, besides the constant introduction of new concepts and approaches to doing things which leads to developers relearning stuff.

Revisiting, React becomes a good choice for saving some time if an interface is wanted to be created as fast as possible, with maximum interactivity. However, if developers are not experienced enough with JavaScript or are not committed to learning it, then an alternative must be chosen.

2.4.3 Vue.js

Vue.js is an open-source library for building interactive web interfaces [58]. To become a framework other libraries are needed, just as in React. It was designed by Evan You and emerged from his work at Google where he was making prototypes with Angular. He extracted what he found good on Angular and created something lighter.

Since then, Vue.js has been learning from the mistakes and successes of React and Angular [59], and it possesses the best of both, such as virtual DOM, component-based architecture, and two-way data binding features.

The main idea behind the Vue development is to deliver a much simpler concept of a framework, providing the bare minimum of what would be expected from a JavaScript framework [60]. Vue.js applications can start small and simple, and then build bigger and better, while another framework would make complexity the default.

From using Vue.js, the following advantages can be identified:

- **Easy Learning Curve and simple applications builiding** due to the offering of basic frontend development tools such as JavaScript, CSS, and HTML. Thus, learning Vue.js doesn't require a lot of background, and applications are designed from scratch. And, if needed, complexity can be added;
- **Fast Renderization** due to the lightweight nature of Vue.js and the virtual DOM functionality;
- **Extensive and comprehensive documentation** that provides instructions on basic and usually needed features such as routing and state management.

However, it presents the following disadvantages:

- **Excessive flexibility** becomes a problem because developers are allowed to really start from scratch and then, the incremental implementation of new features may cause confusion as errors and irregularities start to evolve on larger projects;
- **Too Limited** because it does not have as many plug-ins or components as other frameworks of its kind, even though it has official libraries and communities;
- **Too New** which leads to a community that is still small and thereby support for quick error solving may be scarce;
- **Language barriers** since many of its related projects and users are non-English speakers, and therefore materials are not understandable [61].

In the end, Vue is seen as a beginner-friendly framework, adequate for building prototypes, or creating applications that begin simple and may grow bigger and more complex. However, it is not suited for larger projects since it is relatively new and there may be a lack of support, and quick problem-solving.

2.4.4 Svelte

Svelte is an open-source component-based Typescript-written JavaScript framework. In fact, it is neither a framework nor a library, but a compiler [62].

Svelte is known as a lightweight front-end development alternative, which allows developers to create projects with much less coding when compared to other frameworks. It is designed to do as much of the work as it can at build time, rather than in the browser [63].

Some of the important features of Svelte are the absence of a virtual DOM and the modularity promoted in the coding process allowing to group different components and isolate the template, logic, and view. This modularity allows accessing variables straight from markup, making the whole development navigation easier.

Svelte also enables writing boilerplate-free code, meaning verbosity is not present. Thereby code becomes much cleaner, allowing easier and faster creation of components. Later, at the build step, the code of those components is processed by the compiler into lightweight standalone modules in vanilla JavaScript (i.e. framework-less). These modules are then precisely integrated into the DOM when the state changes. As a consequence, Svelte does not require high browser processing and there is no need to spend resources on building a virtual DOM.

The main advantages of using Svelte are:

- **Less code:** since there are no extra complexities to be added related to verbosity or standards, allowing for a more focused development on business logic;
- **Reactivity and user experience:** since, in the first place, it does not use a virtual DOM that slows the process of updating the interface. Secondly, because it runs at build time, converting components into highly efficient imperative code, in vanilla JavaScript, that surgically updates the DOM. This allows for an application even faster, which consequently improves the user experience and engagement [64];
- **Integration and interoperability:** as Svelte is simply a compiler, working at the component level. So, even if a project is using Vue or React, it is simple to integrate Svelte components with it [65].

As limitations there are the following:

- **Development support:** because the community is still small which leads to less aiding materials being available;
- **Debugging and Testing:** since there is an absence and lack of mature tools that can help trace and solve errors.

So, Svelte should be considered as an option for small projects as it is suited for beginner front-end developers due to its simple coding and needless of manipulating the DOM, which allows the creation of a fast and reactive product. Nonetheless, Svelte may not be considered for larger projects because there is low support for development either provided by tooling or the community.

2.4.5 jQuery

Although considered a framework, jQuery is an open-source, easy-to-use, cross-platform, and feature-rich JavaScript library. Designed to simplify the client-side scripting of HTML, it makes HTML document traversal and manipulation, animation, event handling, and AJAX very simple with an easy-to-use Application Programming Interface (API) compatible with multiple types of browsers [66].

As jQuery is designed to minimize the time and effort-consuming JavaScript coding task, it offers multiple built-in options for dealing with event handling. This leads to shorter fragments of code, that otherwise would be large, and thereby easy to handle and integrate into any logic part of an application. Released in 2006, jQuery is still being used nowadays [67] and, consequently, its community is huge, experienced and a lot of support can be found.

Also, jQuery is superior when it comes to handling multi-browser support, making it a very attractive option to frontend engineers who do not want to worry about potential incompatibilities between browsers.

So, jQuery advantages are:

- **Browser compatibility:** since jQuery offers solutions that promote a correct functioning of a web page across different browsers;
- **Huge and mature community:** that provides a lot of materials that can be used to learn and quickly solve problems that may arise;
- **Simplicity:** due to its incorporated options which allow shorter codes. With its open coding standards and simple syntax, web designers can shorten the time that it takes to deploy a site or application [68].

However, it also presents some drawbacks:

- **Heavy library:** containing all of its DOM, events, effects, and AJAX components which affect the overall performance of a web application [69];
- **DOM APIs are considered outdated:** as modern browsers can now do the same work, but much more quickly;
- **Absence of a data layer:** which makes direct DOM access mandatory to manipulate it. That increases the complexity of the process when updating the view.

Considering its benefits and worst aspects, jQuery seems to be an efficient tool for developing frontends that require cross-browser support. Besides, its set of provided functionalities facilitates the creation of web pages and enables the delivery of interactive ones even in 2021. However, unlike many modern frameworks, jQuery lacks a data layer, so the DOM access is always direct. Consequently, the process of updating the view becomes more complicated and the code starts to become larger, decreasing the webpage's performance. That's why if the goal is to build a complex UI, another framework should be a better option.

2.4.6 Ember.js

Ember.js is an open-source and free JavaScript client-side framework used for developing web applications. It allows building client-side JavaScript applications by providing a complete solution that contains data management and an application flow [70].

Ember.js is component-based and well-organized. Some rules must be followed in the development process which end up restricting and guiding it. Thus, a standardized structure can be achieved by not allowing much flexibility.

One of its features is the two-way data binding which allows synchronization between the view and model in real-time. It also has a templating mechanism that allows developers to minimize the total amount of code that needs to be written. Moreover, it incorporates some powerful features and components which can be extended by plenty of available plugins and tools developed by its substantial community.

There are 5 core concepts on Ember.js, depicted by Fig. 2.10, which are its fundamental building blocks: routing, models, services, controllers/templates, and components [71]:

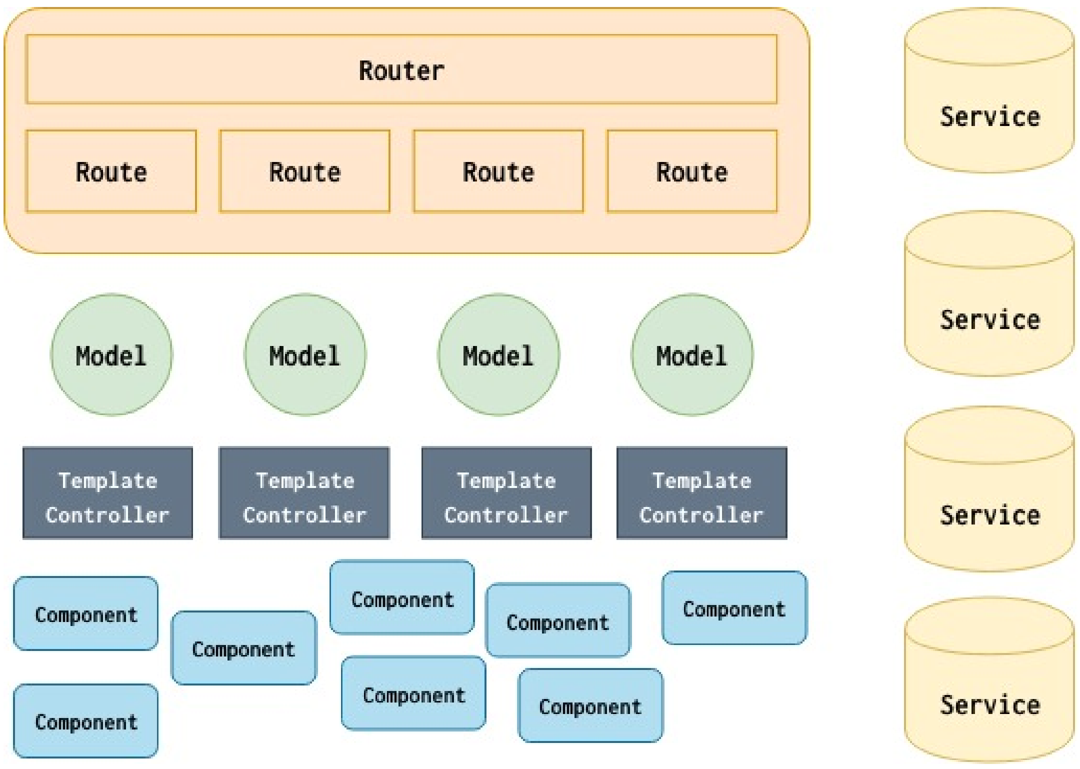


Figure 2.10: Ember's core building blocks (from [71]).

- **URLs and Routing:** routes are features used by Ember.js which allow driving the application state from the current URL. The *Router*, from Fig. 2.10, is responsible for mapping the current URL to one or more route handlers. Then, those route handlers can render a template, load models for templates, change models or redirect to new routes [72];
- **Models:** A model is a representation of an entity. It represents the structure of a data object, defining the attributes, relationships, and behavior;
- **Services:** These are objects that live for the duration of the application. Their function is to provide services to other objects. Whenever a service is needed to extend a component's behavior, it should be injected to be made available;

- **Controllers and Templates:** Templates are basically the HTML pages. They define views that can use properties of model instances in the display. Controllers are objects responsible for defining UI logic around the model and other stuff like query parameters. That logic can then be used in the Templates to customize the view;
- **Components:** A component is something like an HTML tag, but customized, meaning it is formed by HTML elements or even other components. It is like controllers and templates, but reusable.

From Ember we can take the following advantages:

- **Ready-made structure of the application:** provided by the framework which imposes a general application structure and organization for its users. The goal is to prevent developers from making mistakes that would only confuse their apps [73];
- **Stability:** because the framework's constant improvement is the ultimate aim and backward compatibility is ensured. The upgrade procedure is simple, consisting first in alerting the user about forthcoming changes via deprecation warnings. Furthermore, Ember intends to release a new, stable version every six weeks [74];
- **Support:** provided by its considerable and active community and by the complete Ember's official documentation;
- **Built-in testing and debugging tools:** that supports the development activity aiding with error fixing;
- **Two-way data binding** that allows synchronizing the view with the model, and thus avoiding writing extra code in situations, such as forms, in which that synchronization is necessary.

Yet, it has some drawbacks like:

- **Conventional and rigid structure** which difficult the learning and development process [75];
- **Heavyweight** which affects the overall performance of an application. In fact, it is one the of heaviest frameworks [76].

So, Ember.js becomes ideal when the goal is to create complex and feature-rich web applications. Nonetheless, if the project or the team is small, Ember.js may be too complex and overwhelming.

2.4.7 Semantic UI

Semantic UI is a free, open-source front-end development framework designed for theming, that is a CSS framework. It contains pre-built semantic components that help create beautiful and responsive layouts using human-friendly HTML [77].

The framework utilizes concise HTML, treating words and classes as exchangeable concepts. Classes use syntax from natural languages like noun/modifier relationships, word order, and plurality to link concepts intuitively [78]. Also, Semantic UI allows integration with React, Angular, Meteor, and Ember.

The goal of this framework is to ease the frontend development by providing a human-friendly HTML syntax (semantic method). It delivers a vast set of tools for configuring themes and CSS.

This framework has the following advantages:

- **Self-explanatory code** due to the syntax used that is close to a human natural language;
- **Rich and responsive components** from the huge set provided, which results in numerous alternatives for approaching a modern and responsive design;
- **Integration** with well-known and used frameworks such as React and Angular;
- **Documentation** that is well organized and structured, with multiple examples.

However, it presents the following drawbacks:

- **Relatively small community** which may result in less support for fixing or solving issues;
- **JavaScript knowledge** since many of its features are JavaScript dependent, the developer needs to be well familiarized with the language to solve issues [79].

In the end, Semantic UI is a framework that allows delivering a fast, elegant, and responsive design of user interfaces. Nonetheless, it may not be good to use it with inexperienced JavaScript developers, because qualifications are required for implementing customizations in the application without depending on the ready-made functions.

2.4.8 Frameworks decision

After studying and analyzing the benefits and drawbacks of the just presented set of front-end frameworks, a choice must be made on which could be more helpful to use in the development of the product. And considering the scope of this dissertation some aspects influence the frameworks decision:

- **Support** materials that provide sources for quick error solving, and also for learning purposes;
- **Experience** with the framework, which turns into a development boost in the initial phase of the frontend development, since there is past knowledge. As a consequence, an initial learning phase can be skipped;
- **Testing and debugging** mature tools which aid in error solving and later at the validation of the product;
- **Simplicity** which translates into less code, easier learnability, and faster development activities;
- **Reactivity and Responsiveness** capabilities, since these are the properties that most define how usable a UI is. Besides, the decision support tool that is going to be created should achieve just that, good usability, so that a tool capable of aiding on system's management activity is built;
- **Structured** code that enables separating components and concerns, making the maintainability task a lot easier.

In Table 2.1, a match between each of the frontend frameworks and the decision-influencing aspects is shown.

Table 2.1: Comparison of frontend frameworks.

Factors	**Frameworks**						
	Angular	***React***	***Vue***	***Svelte***	***jQuery***	***Ember***	***Semantic UI***
Support	X	X			X	X	
Experience		X					
Testing and debugging	X	X	X		X	X	-
Simplicity		X	X	X			X
Reactivity and Responsiveness	X	X	X	X		X	X
Structured	X	X	X	X	X	X	X

As it can be seen, React is the single framework that fulfills all the aspects meant to compare among the frameworks. Therefore it becomes undoubtedly the framework choice for implementing the tool of this dissertation.

When it comes to Angular, simplicity is the strongest factor for refusal, once it would slower development and require much more coding and time to learn.

As for Vue, it is still a young framework so, even though it has official libraries, the community is still small, and thereby support is affected.

In its turn, Svelte could be an interesting choice due to the compiler feature which speeds up a lot of the renderization processes. However, the community is still small and, naturally, the available support is affected. Besides, debugging and testing tools haven't reached a significant maturity level.

Relatively to jQuery, it doesn't have a data layer, so accessing the DOM is mandatory. Thereby the process of updating the view becomes way more complicated and thereby simplicity is a downside factor. Moreover, the DOM APIs provided are considered outdated so the performance might not be optimal and, as a consequence, reactivity and responsiveness may be affected.

Respecting Ember, its heaviness may be harmful to the product performance. However, the property that makes it be considered less of an option is its rigid structure of doing things, because it affects the development process time and the simplicity of the framework, making it harder to learn.

As Semantic UI is a CSS framework, it does not make sense to assess it on testing and debugging tools. But for the rest, the only disadvantage presented is on support since it has a relatively small community. Thereby, there may not exist sufficient support for solving issues that might pop. However, Semantic UI is very easy to use since its CSS syntax is very close to natural language. Also, it offers a lot of ready-to-use components and it is a framework that allows building modern and responsive user interfaces easily and quickly. Moreover, it can be integrated with React. For all these reasons it makes sense to use it alongside React to build the frontend product of this dissertation.

Chapter 3

State of The Art

There are multiple trends leading organizations to convert their cloud applications into microservice applications, making them highly scalable and available [80]. Also, service-oriented architecture, like the microservices, has often been used as a mechanism for achieving self-adaptiveness on systems [81].

Self-adaptation is a concept that has been present in other domains such as biology and economics. But in the domain of software, it means that a system monitors itself and the operating environment, dispatching actions when it detects changes that require adaptation [82]. In the Cloud, those adaptations are taken according to the demand and are generally associated with the increase and decrease of resources such as computing power and bandwidth.

In this chapter, more details on what are self-adaptive systems will be presented. Then, some self-adaptive enabling tools will also be presented and described in detail. The first tool, and the one this work is related to, is Trustworthiness Monitoring & Assessment Framework (TMA). Its relation with concepts, previously presented in Chapter 2, like microservices and containers, is also explained. Also, it is made a comparison between all the tools, including TMA, and a discussion on why should TMA be used instead of the others. Finally, details of an experiment performed with a self-adaptation enabling tool provided by Kubernetes are presented.

3.1 Self-Adaptive Systems

Self-adaptive systems can be seen as computing environments capable of self-managing and adapting to changes according to business policies and objectives. They can adapt based on observed or sensed situations in the Information Technology (IT) environment instead of requiring supervision from human resources to perform the tasks. To be considered as self-adaptive, systems possess one or more of the following attributes [83]:

- **Self-configuring:** Components dynamically adjust to changes in the environment following policies provided by IT professionals. Those changes may be e.g., the deployment of new components or removal of existing ones. Dynamic adaptation helps ensure continuous good performance of the IT infrastructure;

- **Self-healing:** Components are capable of detecting system malfunctions and initiating policy-based corrective actions without disrupting the surrounding IT environment. These actions can be altering their state, or executing changes in other environment components. Thus, the whole IT system becomes more resilient;

- **Self-optimizing:** Components can self-optimize to meet end-user or business needs. The tuning actions could mean reallocating resources (e.g., in response to changing workloads) to improve overall utilization, or ensuring that particular business transactions can be quickly completed. This attribute helps to provide a high-quality service for both the system's end-users and a business's customers. Without this property, when assigned computing resources are not being fully consumed, there is no easy way to free excessive allocated resources to be used by lower priority work. In these situations, customers must buy and maintain separate infrastructure for each application to meet that application's most demanding computing needs;

- **Self-protecting:** Components can detect the occurrence of undesired behaviors and take corrective actions to make themselves less vulnerable. Those unpleasant behaviors can include unauthorized access and use, virus infection and proliferation, and denial-of-service attacks. This attribute allows businesses to continuously ensure security and privacy policies. Thus, it becomes possible to automate tasks IT professionals must perform to configure, heal, optimize and protect the IT infrastructure.

Considering these properties, IBM proposed a control loop, denominated **MAPE-K**, for enabling self-adaptation on systems [83]. MAPE-K is an acronym where each letter represents one of the main components of the control loop (M-Monitor, A-Analyze, P-Plan, E-Execute, and K-Knowledge).The general idea consists of the Monitor that collects metrics, the Analyze that processes those metrics to identify and predict system states/behaviors, the Plan that creates plans based on policies and the Analyze's output, the Execute that is responsible for executing the plans, and the Knowledge which hosts data shared by the other components.

Fig. 3.1 presents the proposed architecture for the **MAPE-K** control loop, composed of 5 main components that work together to self-manage a system:

- **Monitor:** Responsible for providing mechanisms that collect, aggregate, filter, and report details (such as metrics and topologies) collected from managed resources;

- **Analyze:** Provides mechanisms that correlate and model complex situations (for example, time-series forecasting and queuing models). These mechanisms allow the system to learn about the IT environment and help predict future situations;

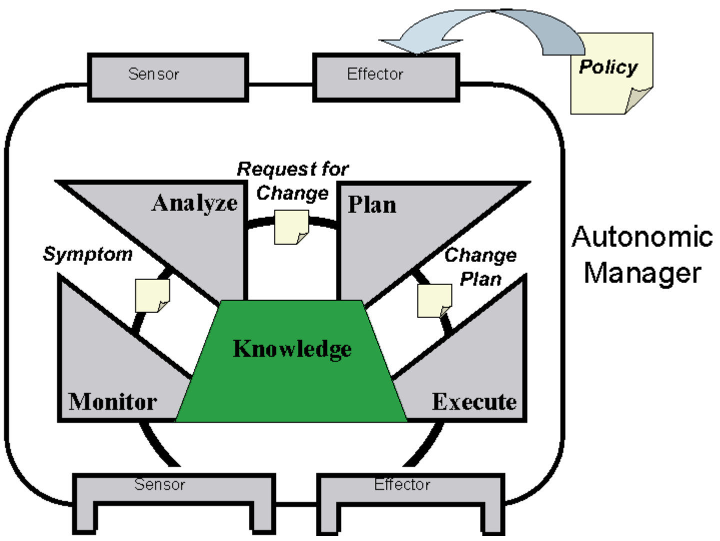


Figure 3.1: IBM proposed architecture for self-adaptive systems (from [83]).

- **Plan:** Provides mechanisms that formulate and create the actions needed to achieve and assure goals and objectives. The planning mechanism uses policy information to guide its functions;
- **Execute:** Responsible for having mechanisms that control and orchestrate the execution of a plan;
- **Knowledge:** A repository accessible by the previous components, where information and data are shared.

Besides these components, there are two more, **sensors** and **effectors**, also known as probes and actuators respectively. The first is responsible for actually collecting raw information from systems states and making them available for the *Monitor* component. Effectors are the ones responsible for actually performing the adaptations. They receive orders from the *Execute* component to assure a plan is properly executed.

The remaining of this section will present systems that either implement the *MAPE-K* control loop or implement any self-adaptive capability.

3.1.1 TMA

TMA is a platform that follows a microservice architecture, implementing a MAPE-K control loop [18]. Functions from that control loop are distributed by the multiple components that compose TMA: *monitor*, *analyze*, *planning*, *execute* and *knowledge*.

The overall functioning of the platform starts with *probes*, which correspond to the MAPE-K's sensors, collecting data from resources and sending it to the *monitor*. Then, the data received by *monitor* is persisted in the database provided by the *knowledge* component. Next, *analyze* processes the data gathered and calculates scores through quality models. These scores are then compared against thresholds by *planning*, which in cases of disrespect generates plans for the system to adapt. *Execute* gets these plans and makes them happen, invoking *actuators*. Basically, it orchestrates the whole adaptation, making the calls, while *actuators* are the ones who actually perform the adaptations.

To assure that communication among the components is resistant to possible failures and handled reliably, a fault-tolerant mechanism is used. For that, Apache Kafka is used and topics are created so that components can publish information to them or subscribe to receive messages.

Each of the components that constitute TMA has a Dockerfile that details every single software package needed to create the respective Docker container image. Those Docker images correspond to containers that are then used in YAML Ain't Markup Language (YAML) files. These files have several specifications which are used by Kubernetes to understand how to deploy and manage the containers. Docker is a technology that eases the process of dealing with containers, from creation to execution phase, while Kubernetes is an open-source platform that allows managing and orchestrating multiple containers inside a network.

Following it is presented the architecture of TMA on Fig. 3.2 and the behavior of the components that represent the MAPE-K control loop are detailed:

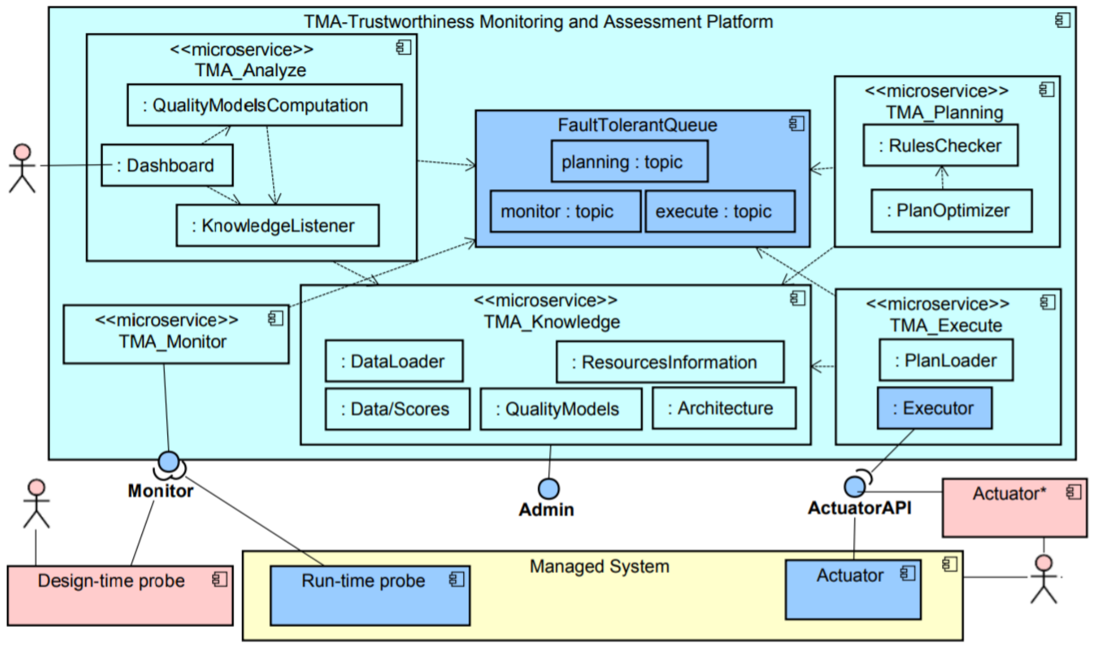


Figure 3.2: TMA architecture (from [18]).

- **Monitor -** Component, deployed using the web microframework Flask, that provides a Representational State Transfer (REST) Application Programming Interface (API) for probes to post observations (JSON messages) from the managed elements. The message contains a data type, that can be a measurement (e.g., memory allocated, CPU usage) or an event (e.g., the scale-up process has started). While measurements are numeric values used to calculate scores based on quality Models (QM), events are occurrences. Additionally to the data type, a probe's message contains the value and time of the observation.

When data from probes is received, it is validated according to a JSON schema. If the schema isn't respected, the message, and consequently the data, is discarded and an error message is returned to the probe.

After verifying the message's schema is valid, the data is enqueued, in a reliable way, at the *FaultToleranteQueue* in a topic named "topic-monitor", to assure it will be later stored in the *Knowledge* component. The persistence of the probed data in *Knowledge* is accomplished through a component called *DataLoader*. That component pulls data from the monitor topic and executes a data normalization process, making it correct for insertion in the Knowledge database.

To ensure secure communication between probes and the *Monitor* component, SSL/TLS encryption is used. Thereby, probes must possess the Monitor's digital certificate to be able to communicate with it;

- **Analyze -** This component is responsible for evaluating the data gathered by *Monitor*, applying quality models. In the decision-making (adaptation) process, it may become necessary to consider several properties of a system. Thus, this component uses quality models which are structured weighted trees of quality attributes. This allows to reason about adaptations on multiple sources of information. Basically, quality models are used to aggregate the measurements from a system and come up with a final score that reflects a system state based on a set of properties (e.g., security and performance). Fig. 3.3 presents an example of a quality model that could be applied.

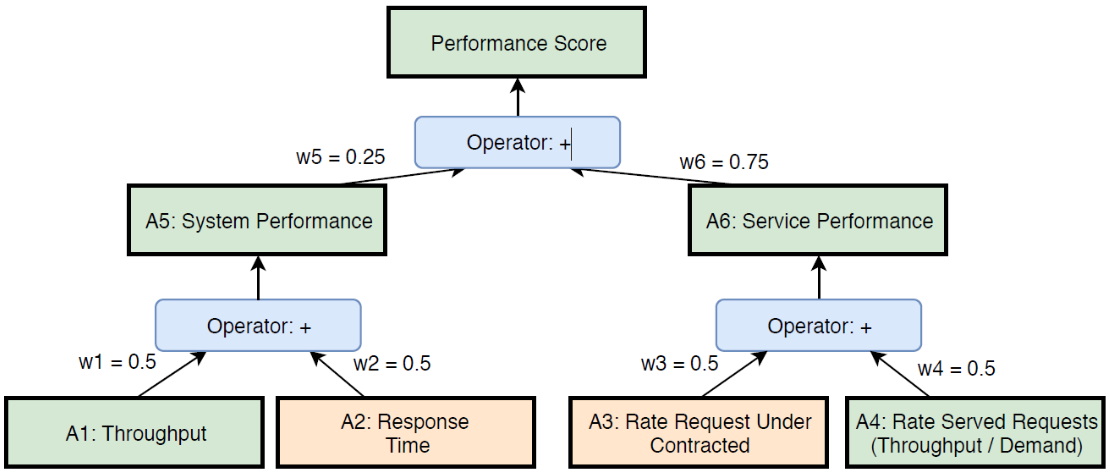


Figure 3.3: Example of a TMA quality model (from [18]).

During execution, *Analyze* aggregates the measurements from a resource, calculates metrics values for each of the nodes that compose the quality model tree, and finally, it outputs a score for the root node metric, made upon multiple sources.

The data collected from probes are read from the Knowledge component and the calculated values are also stored in it. These values are also sent to a topic, called "topic-planning", in the *FaultTolerantQueue* to be consumed by *Planning*;

- **Planning -** This service is responsible for checking the scores calculated by *Analyze*, retrieved from the topic "topic-planning" at the *FaultTolerantQueue*, and compare them against thresholds. When threshold conditions are verified, and an adaptation is needed, *Planning* has to come up with a plan. An adaptation plan is a set of actions that are meant to be executed to assure defined goals.

  There are different adaptation decision approaches, e.g., models, rules/policies, goals, or utility [84]. TMA uses a business rules management system called Drools, which is a Java-based tool, thus following a rule-based approach.

  When an adaptation plan is created, *Execute* is going to be informed by a message sent to the topic "topic-execute" at the FaulTolerantQueue;

- **Execute -** It is the component responsible for invoking the actions of the adaptation plan defined by the *Planning* component, once notified through the topic "topic-execute".

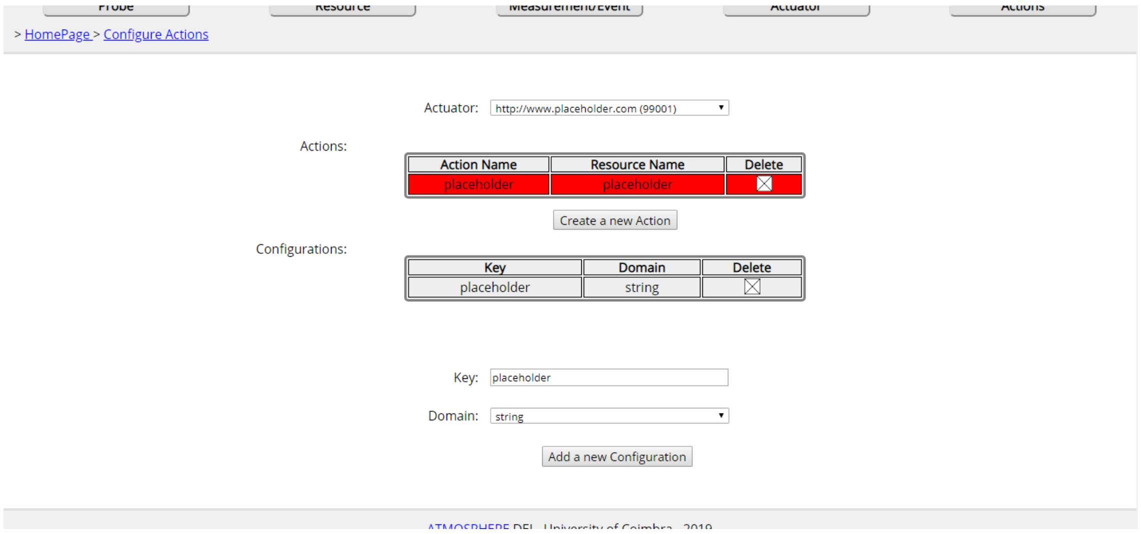
  TMA can interact with the managed element (target of the adaptations) through Actuators. Each actuator provides a REST API which may be invoked by the *Execute* to perform any adaptation. All communication between *Execute* and the *actuators* is done securely, since messages are encrypted using the keys of both the *Execute* and the *actuator* involved;

- **Knowledge-** This is the component responsible for storing all of TMA's data, such as measurements and events collected from probes, quality models definitions, metrics scores, resource information, and adaptation plans.

  Its implementation contains a MySQL DBMS (knowledge database) and a block-storage solution Ceph. There is also a part of *Knowledge*, the *DataLoader*, which is responsible for saving in the database the values collected from probes.

Besides the main components of TMA just presented, there is another service called ***tma-admin-api***, implemented in Java using the Spring framework, which provides a REST API for inserting data into the *Knowledge* database, and other features meant to ease the configuration of the platform, such as generation of keys and getting lists of probes and resources.

There is also another service, a web page developed in Angular, named ***tma-admin-web*** that uses the API component to provide a User Interface (UI) for TMA's administrators. Here there are available functionalities like the generation of keys for actuators, and the creation of TMA's database entities such as resources, descriptions, probes, actions, and actuators. Besides not having a minimal carefully designed aspect, this interface has the single purpose of creating noncomplex TMA configuration data on the database as Fig. 3.4, 3.5, 3.6 and 3.7 depict. As the figures show, clearly this web page was not thought to support the decision process when managing systems since it does not have any feature in that sense.



> HomePage > Configure Actions

Actuator: http://www.placeholder.com (99001)

Actions:

Action Name	Resource Name	Delete
placeholder	placeholder	☒

Create a new Action

Configurations:

Key	Domain	Delete
placeholder	string	☒

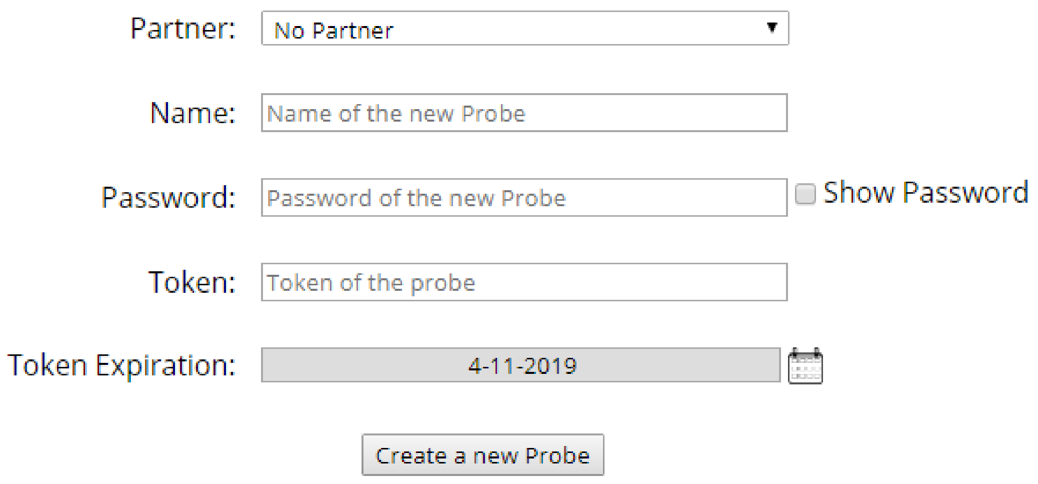
Key: placeholder

Domain: string

Add a new Configuration

ATMOSPHERE DEI - University of Coimbra - 2019

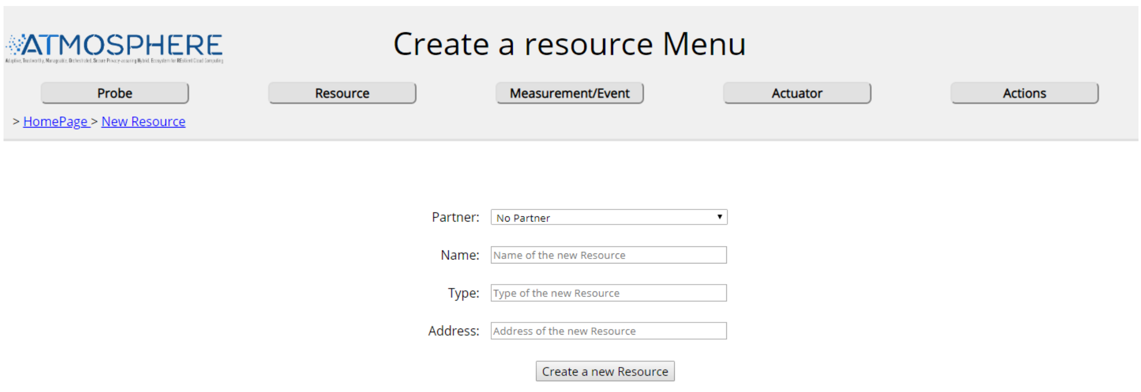
Figure 3.4: tma-admin-web's create action page (from [85]).



Partner:	No Partner
Name:	Name of the new Probe
Password:	Password of the new Probe ☐ Show Password
Token:	Token of the probe
Token Expiration:	4-11-2019

Create a new Probe

Figure 3.5: tma-admin-web's create probe page (from [85]).



> HomePage > New Resource

Partner:	No Partner
Name:	Name of the new Resource
Type:	Type of the new Resource
Address:	Address of the new Resource

Create a new Resource

Figure 3.6: tma-admin-web's create resource page (from [85]).

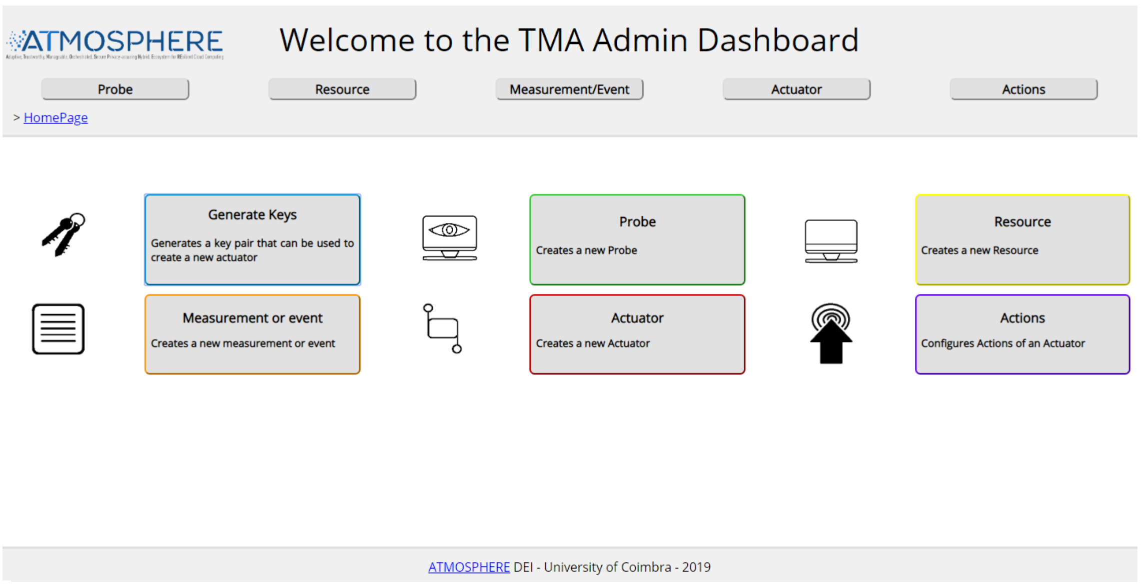

ATMOSPHERE
Welcome to the TMA Admin Dashboard
Probe
Resource
Measurement/Event
Actuator
Actions
> HomePage
Generate Keys
Generates a key pair that can be used to create a new actuator
Probe
Creates a new Probe
Resource
Creates a new Resource
Measurement or event
Creates a new measurement or event
Actuator
Creates a new Actuator
Actions
Configures Actions of an Actuator
ATMOSPHERE DEI - University of Coimbra - 2019

Figure 3.7: tma-admin-web's menu page (from [85]).

For implementing **probes** and **actuators** there are base scripts in java, python, and C#, which reflect how they should behave and be constructed. These base scripts can be used as starting points to further expand their behaviors and implement own probes and actuators. Also, there are some developed probes and actuators ready to use if they fit the needs.

TMA relation with Microservices and Containers

TMA's architecture follows the microservices principles, separating the systems monitoring and adaptation tasks it performs across multiple and smaller components. Essentially, each of those components is a container responsible for performing a related set of TMA's functions. For example, the Analyze component deals with the data processing and calculation of metrics values, while Planning is only responsible for applying rules and triggering adaptation plans. Also, as TMA is split into different services, each can be scaled individually and independently from the others. Meaning that if a sector of the management loop is more loaded than the others, it can be scaled to deal with demand.

In its implementation, TMA makes use of both container technologies, Docker and Kubernetes.

First, Docker is initially used to build container images for each of the microservices composing TMA. There is a Dockerfile for each service that specifies how the image is built, including all the code and necessary dependencies of that component. Those Dockerfiles are run to consequently build the images. In case changes are made to a component's code, its Dockerfile needs to be rerun to rebuild and update the container image.

After generating the images, they can be shared on Docker Hub which is a repository of public and private images provided by Docker. The advantage of using the DockerHub is to make the *TMA*'s components images available anywhere so that separated deployments of *TMA* can be maintained elsewhere.

Finally, Kubernetes is used to deploy, run and manage the components Docker images in a cluster. For that, there are YAML files that hold descriptions of how those images should be run, restarting policies, available resources, and other kinds of configuring information.

3.1.2 Hogna

Hogna is a platform that allows deploying self-managing web applications on the cloud. To begin, it enables deploying applications through the automation of a series of steps, such as starting and configuring instances. After that, it provides continuous monitoring of the deployment where the data collected is analyzed according to a performance model. Finally, adaptation plans are created and executed. Furthermore, the components involved in the process of deployment to the adaptation can be customized to fit the needs [86].

Hogna authors claim that the platform provides the following set of features:

- Automatic setup of a topology at a cloud provider, by specifying and describing the cloud network layout at a file descriptor;
- A monitoring component that manages all the monitors and extracts metrics from them periodically, thereby allowing continuous monitoring of the deployed topology. This component may extract metrics from a cloud-specific monitoring API or agents attached to topology instances;
- A performance model upon which metrics collected are analyzed and plans formulated;
- A mechanism that allows inserting logic into the analyzing and planning phases.

This platform's architecture, represented by Fig. 3.8, can be seen as 2 subsystems. The first is composed of the monitoring engine, analytical performance model, execution engine, and the management logic for the deployed application. The second subsystem is optional and constituted of components found useful in supporting testing activities.

To understand Hogna's functioning, details on the primary subsystem components are given next:

- **Management Component:** This is the core component of Hogna, where a MAPE control loop is applied. Periodically, recently collected metrics are extracted. Then, those metrics along with a performance model are passed to the analyzer. The results from the analyzer are then sent to the planner which will create a set of actions that need to be performed so that identified issues can be corrected. The plan is later sent to the execution engine which will be in charge of performing it;

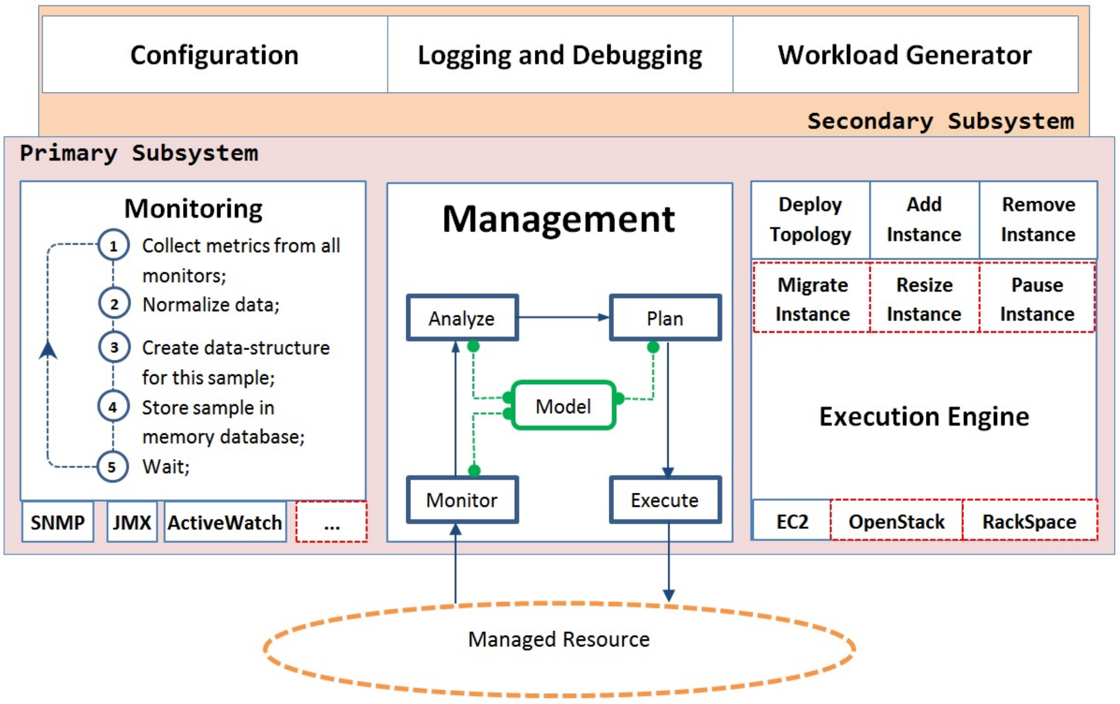


Figure 3.8: Hogna architecture (from [86]).

- **Performance model**: This is a built-in element provided by Hogna. Based on the definition of this model, the analyzer and planner make their decisions;

- **Monitoring Component**: This component manages a set of monitors, and it is responsible for continuously extracting metrics from them. It has its own internal loop, composed of the steps described in Fig. 3.8, which starts with the request of metrics from the set of monitors. Once data arrives, it is normalized and then inserted into a database.

  The set of monitors is loaded automatically from the topology configuration file and each monitor is associated with an instance in the topology;

- **Analyze Component**: This component is responsible for evaluating the state of a managed resource. Based on the metrics received, the topology description, and the performance model, it calculates a system status. That status is then communicated to the planner.

  From source, Hogna provides an analyzer that operates using threshold rules to decide if the system is or isn't overloaded or underloaded;

- **Planning Component**: This unit is responsible for coming up with a sequence of actions that will resolve issues identified by the analyzer. That sequence of actions constitutes a plan that will later be sent to the execution engine.

  From source, Hogna only provides add and remove instances operations;

- **Execution engine**: This is the component responsible for performing the theoretical plans designed by the planner.

3.1.3 Lotus@Runtime

Lotus@Runtime is an extensible tool that uses models at runtime to monitor and verify self-adaptive systems [87].

This tool maintains a probabilistic system model which defines the probabilities of occurrence of certain system actions. The tool then monitors the execution traces of a self-adaptive system and updates the probabilities of the model.

During runtime, the probabilistic model is checked against some system actions to verify that properties still hold. If a property is violated according to the probabilistic model, the self-adaptive system can be informed by a notification mechanism provided by Lotus@Runtime.

Thus, Lotus@Runtime provides real-time monitoring of applications through the execution traces generated by a self-adaptive system and allows verifying during runtime certain system properties.

Lotus@Runtime architecture can be seen on Fig. 3.9 and it is composed of the following elements:

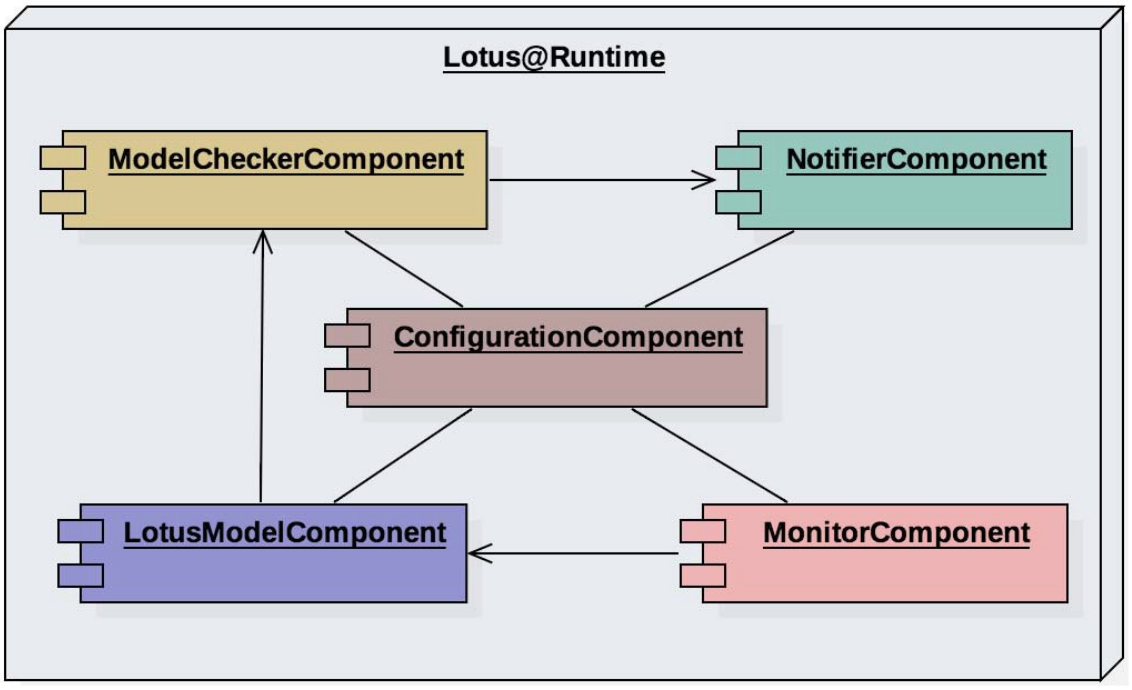


Figure 3.9: Lotus@Runtime architecture (from [87]).

- **MonitorComponent**: This component is responsible for monitoring a system by reading its generated execution traces. A trace is a sequence of visible actions which represent events and a set of traces is a log. The monitoring operation is performed by reading the log file, where each line represents an execution trace.

  This component can monitor the traces either periodically in a time defined by the user or by reading the log file when it changes;

- **LotusModelComponent**: This element is responsible for maintaining a model updated from the information collected by the *MonitorComponent*. It also provides services that allow other components to receive the updated model;
- **ModelCheckerComponent**: As the probabilistic model is updated, this component is responsible for performing runtime verifications so that it becomes possible to find violations of properties defined by the user. In case the result violates a property, the notification service should be invoked;
- **NotifierComponent**: This component is in charge of publishing properties violations in an event bus, which can later be read and used in the planning phase of a self-adaptive system to design adaptation plans;
- **ConfigurationComponent**: This component is simply in charge of hosting, in a centralized way, configuration parameters that will be read from other components. Those parameters are configuration file name, log file path, model file path, optionally the time to check for new traces, and the list of properties to be verified at runtime.

How Lotus@Runtime allows for self-adaptiveness is through the flow represented by Fig. 3.10. It all begins with the monitoring process that collects traces from the system. Then, the information collected is used in the updating process to update the probabilistic model. After that, in the verification phase, the probabilistic model is checked against properties defined by the user.

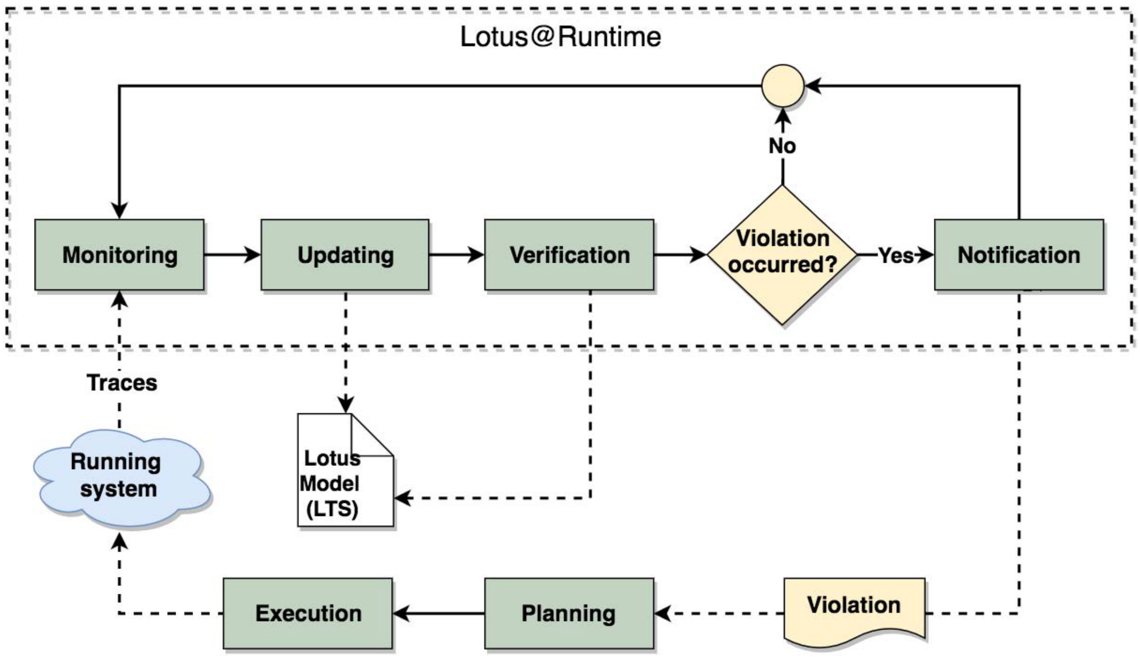


Figure 3.10: Self-adaptive systems with Lotus@Runtime (from [87]).

If a property is violated, the next step is the notification process that will transmit the violation data to an external component. This component that receives the violation details will be in charge of the planning phase of the self-adaptive system. Then, another external component will be responsible for performing, at the execution phase, the plan designed.

3.1.4 Summary

After studying and analyzing a set of different tools that provide self-adaptation capabilities to systems, following there is a discussion and a comparison to justify the use of TMA. To reason about this choice the following properties were considered:

- **Complete control loop:** the tool implements a control loop that covers the whole systems management activity from monitoring to adaptation;
- **Environment flexibility:** the tool can be applied to any system within any type of environment being it cloud or not, public or private;
- **Monitoring flexibility:** data collected from systems when monitoring can be of any type;
- **Unlimited applicability:** a single instance of the tool can provide self-adaptation to a variety of systems or several components of the same system. This means that the tool is not restricted to a maximum number of systems or any other restriction.

In Table 3.1, a match between each of the self-adaptive tools and the properties considered is shown.

Table 3.1: Comparison of self-adaptation tools.

Properties	**Tools**		
	TMA	***Hogna***	***Lotus@Runtime***
Complete control loop	X	X	
Environment flexibility	X	X	X
Monitoring flexibility	X	X	X
Unlimited applicability	X		

As Table 3.1 depicts, TMA is the only tool that fulfills all the properties considered for the comparison between self-adaptive tools.

Concerning TMA, it has a complete control loop by implementing the logic that crosses the whole management activity lifecycle (monitor, analyze, plan and execute). Also, TMA has no restrictions in the types of environment it can be used. And, when monitoring, any type of data can be collected from systems being the only requirement to implement that collection on probes. Ultimately, it also has no restrictions on the number of systems it can manage at the same time by applying different quality models to different systems or parts of the same system.

Hogna, like TMA, implements a complete control loop, has flexibility in the monitoring data types, and can be applied to any environment. However, it is tied to a topology defined in a configuration file, meaning that it can not be applied to other systems at the same time. Also, with Hogna, the same performance model is applied to the whole setup, which means that parts of the system can't be individually managed, only the setup as a whole.

Looking at Lotus@Runtime, it presents benefits on the environment applicability since there are no restrictions. Also, its monitoring flexibility is attractive as it operates by reading traces from the monitored system and those traces could be anything the user wanted. However, Lotus@Runtime presents downsides respecting the control loop and the limits where it is applicable. To begin with, this tool does not implement the logic from the planning and execution phases of the management control loop. Instead, it leaves that implementation to be performed by the user. Finally, it seems to be applicable to a single system, meaning that another instance of the tool has to be run for each new system being monitored.

As analyzed and mentioned above, TMA is the tool that most suits the needs. Consequently, it makes sense to choose it as the self-adaptation enabling tool of this work.

3.2 Experiment with Kubernetes Horizontal Pod AutoScaler (HPA)

In the context of this dissertation, an experiment was made to study the elasticity provided by a Kubernetes self-adaptation enabling tool, HPA, for a microservice application. Elasticity is the degree to which a system can autonomously adapt to workload changes by adjusting the resources consumed to the current demand as closely as possible [88].

Kubernetes provides a feature for run-time demands called HPA. It can dispatch scaling actions based on CPU consumption, memory utilization, or custom metrics [89] collected from pods by a tool called `metrics-server` (`https://github.com/kubernetes-sigs/metrics-server`). Scaling decisions are made by HPA upon collected metrics through a formula and defined thresholds. Scale-ups are performed immediately, while scale-downs take longer by considering the highest value calculated for the number of pod replicas in the last N seconds (**stabilization period**). By default, N is 300 seconds. This is to avoid destroying pods when the load decreases transiently.

3.2.1 Experimental Methodology

This section presents the steps followed in the experiment to evaluate HPA on scaling the microservice application TeaStore. The goal was to assess the elasticity of Kubernetes HPA when applied to modern microservice applications. For that, it was selected metrics that demonstrate the level of throughput growth, and it was established a relation between it and the extra resources consumed when the demand increases.

The followed methodology consists of 5 steps, as illustrated in Figure 3.11, and they are described in the following sections.

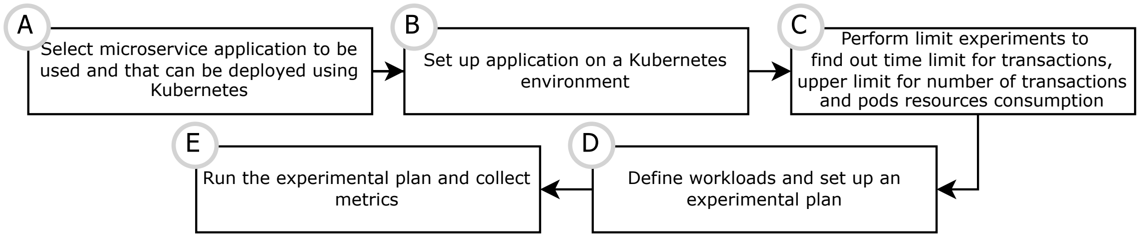

A
Select microservice application to be used and that can be deployed using Kubernetes
B
Set up application on a Kubernetes environment
C
Perform limit experiments to find out time limit for transactions, upper limit for number of transactions and pods resources consumption
D
Define workloads and set up an experimental plan
E
Run the experimental plan and collect metrics

Figure 3.11: Overview of the experimental methodology followed in this study.

A - Select Microservice Application

Considering the objective to assess Kubernetes HPA elasticity, we started by choosing a representative application that can also be deployed into a Kubernetes cluster. We decided on using TeaStore [90], as it can be deployed using Kubernetes, besides being used by several studies about microservices and scaling approaches (*e.g.,* [91], [92]). **TeaStore** is an online store for tea and tea-related utilities [90]. It has been created to represent a modern and flexible microservice application so that researchers can use it to simulate a real-world application.

TeaStore is composed of six services: WebUI, Image, Auth, Recommender, Persistence, and the Registry service that interact with each other through `REST` operations. Through **Registry**, the other services register themselves to make them known and accessible by each other. The Registry is kept updated through heartbeat messages. Besides the Registry service, the **WebUI** is responsible for providing the end-user frontend interface, making it the entry point for the workload requests. **Image** processes images and returns them to *WebUI* to generate the pages to be returned to users. **Auth** deals with verification of login and session data of users. **Recommender** is responsible for providing recommendations of products based on customer behavior. Finally, **Persistence** is responsible for all the accesses to the relational database of TeaStore. The database stores all the information needed by TeaStore, such as users, products, and orders. TeaStore's architecture can be found in detail in [90].

When TeaStore is deployed on Kubernetes using the default configuration, seven pods are created. They are *Auth*, *Db*, *Image*, *Persistence*, *Recommender*, *Registry* and *WebUI*. The names identify the service they represent.

B - Kubernetes Setup with Microservice Application

The second step consists of creating a Kubernetes cluster to deploy the microservice application. For that, it was set up a cluster composed of a `control plane` (4 vCPU and 16GB RAM) and three `worker` machines (4 vCPU, 16GB/16GB/12GB RAM respectively).

Then, it was defined the limits that each service's pod could consume. These limits (specified using the *Kubernetes units*) were distributed equally across the seven pods that compose TeaStore, where limits of *1000m* for CPU and *4Gi* for memory were defined. The CPU unit ***m*** represents the unit ***millicore***. For example, ***1000m*** indicates the consumption of the whole (***100%***) processing time of a single CPU.

C - Define the workload limits

Preliminary experiments were performed to discover and define important aspects of the experiment. With the objective of performing an experiment as real as possible, and as TeaStore represents an e-commerce website, it was searched on the literature for information on how the requests of an e-commerce website are distributed (database writes and reads). *Zhang et al.* presented different workloads, which vary on the distributions of the type of interactions that are performed [93]. Those workload distributions have as a baseline the TPC-W benchmark, which simulates the database transactions of an e-commerce website. At last, it was decided to follow the distribution composed of 95% of interactions that only perform read operations and 5% of interactions that perform both read and write operations.

TeaStore provides two different transactions: *i) browse* (containing only read operations), and *ii) buy* (containing both read and write operations). Hence, we defined the proportions of the transactions to be submitted on the experiments as 95% of browse transactions and the remaining 5% of buy transactions. The requests that compose each of these transactions are available at `https://github.com/DescartesResearch/TeaStore/tree/master/examples/httploadgenerator`. The **buy** transaction is composed of 13 requests, from which 11 are the same as the ones that form the **browse** transaction. The other 2 requests are the ones that write in the database. After this, it was investigated the time a transaction needs to be completed, the maximum number of transactions a **default deployment of TeaStore** (1 pod per service) can handle, and the number of resources consumed by the pods. It was found that when the system starts becoming overloaded, about 75% of the transactions will last at most 3 seconds. Thereby, we defined that transactions would take a maximum of 3 seconds.

Teastore provides workload emulators, however they can not control the pace at which transactions are started. Therefore, a throttle mechanism was created to make sure each user performs transactions at a constant pace, in our case, every 3 seconds. We also adjusted the timeouts of the requests that compose the transaction by a value proportional to the division of those 3 seconds by the number of requests. It was observed that the default deployment could handle up to 35 simultaneous users.

After these preliminary experiments, we decided to reassign resources to pods as it was observed a significantly much lower consumption than the provided limits. Hence, for our final experiments (step E of Fig. 3.11), it was defined the default deployment to be used. Each pod starts with one replica, and most of them (*Auth*, *Db*, *Image*, *Persistence*, and *WebUI*) have 1000m of CPU and 2Gi of Memory each, while *Recommender* and *Registry* have 250m of CPU, and 1Gi of Memory each.

D - Define Experiment Configuration

To evaluate TeaStore, we defined a workload with some abrupt load variations intending to represent a possible realistic load of an e-commerce website. Fig. 3.12 presents this workload, with the number of simultaneous users per experiment slot, where each slot lasts for 5 mins. The workload simulates the traffic behavior of a real website and has a slight variation in the number of website visitors, resulting in the peaks and valleys that can be observed. Also, to represent periods of high load, it was designed the period from the slots 11 to slot 14, with abrupt transitions in the number of simultaneous users. Each user starts a new transaction every 3 seconds.

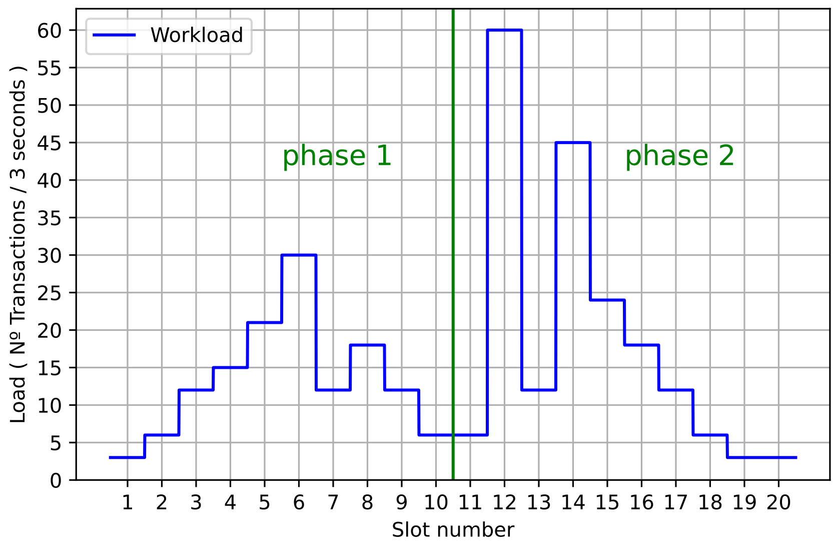


Figure 3.12: Workload divided in 2 main phases along with its slots identification.

The workload is divided into 2 phases, in which the phase *1* has a lower demand (Slots 1 to 10), while the phase *2* has a higher one (Slots 11 to 20). Lower demand means that the maximum number of simultaneous users is 35, which is the limit of load found for the default deployment. Fig. 3.12 shows the workload divided into phases.

Based on the analysis of the measurements of all services of TeaStore in the previous step, we decided to have two services (*Persistence* and *WebUI*) managed by HPA, as both consume almost all CPU that they have available when 35 users are performing actions at the same time. The other services were not consuming half of the resources that were available to them. Using HPA, both *Persistence* and *WebUI* services can vary between 1 and 4 replicas (pods), and the threshold that we used to trigger the adaptation was 80% of CPU usage.

To evaluate the workload and its phases, two types of metrics are used: one is *performance metrics*, and the rate of successful *requests* (`REST` operations submitted to TeaStore) and *transactions* (set of requests that compose a complete activity in TeaStore, such as buying a product) are used. The other type is *resource allocation metrics*, and we use the ratio (indicated by ***R*** in Equation (3.1)) between the **increased throughput** and the **additional resources consumed** when **using HPA**. If R is higher than 1, then the throughput growth is larger than the resource growth. If less than 1, then the reverse happens. If equal to 1, then both grow in the same proportion.

$$R = \frac{ThroughputSpeedup}{ResourceIncrease} \tag{3.1}$$

We decided to estimate the number of resources consumed by a higher value than the actual consumption. For that, in each slot, the highest registered number of pod replicas is considered constant. The *increased throughput rate* (**Throughput-Speedup** (3.2)) is calculated based on the performance metric *Successful Transactions rate*. Dividing this metric from the experiments with HPA by the experiments without, we calculate the *ThroughputSpeedup*.

$$ThroughputSpeedup = \frac{throughputWithHPA}{throughputWithoutHPA} \tag{3.2}$$

The *additional resources consumed rate* only considers the CPU usage (**ResourceIncrease** (3.3)), as we noticed in the previous step (described by Section 3.2.1) that memory consumption barely increases. On the contrary, CPU consumption quickly increases until it reaches its allocated maximum.

ResourceIncrease is calculated by dividing the number of resources consumed on the experiments with HPA by the experiments without. The numerator is given by the sum of each slot's resource consumption. The resource consumption of a slot is given by the sum of each service's resource consumption within that slot (number of pod replicas x pod's cpu). The denominator has an equivalent expression to the numerator however, as adaptation is not used, the number of pod replicas is constant across the experiment. Thereby its value can be calculated through the product between the number of slots (**S**) and the default deployment resource consumption (**TotalConstantResources**), which is equal to the sum of each pod's CPU.

$$ResourceIncrease = \frac{\sum_{s=1}^{S} \sum_{pod}^{P} \#Replicas_s \times CPU_{pod}}{S * TotalConstantResources} \tag{3.3}$$

E - Run Experiments and Analyze Results

Finally, we ran the experiments and collected the respective metrics as planned. In each experiment, a warm-up period of 20 minutes (same period as used by *Eismann et al.* [91]) at a constant load of 5 *Transactions* / 3 *seconds* is run before the predefined workload. For each configuration, we have 30 runs of 2 hours (20 minutes warm-up + 100 minutes of actual workload time). Each configuration is defined by its auto-scaling ability (with or without HPA).

3.2.2 Results and Discussion

In this section it is presented the results and discussion for both performance and resource consumption metrics. The results do not include the warmup period present at the beginning of each run. Also, all the results in this section are the mean values considering all experiment runs, including tables and figures. In our case, all the 30 runs without HPA, and 30 runs with HPA.

Results for Performance Metrics

Table 3.2 shows the results for the complete experiment. **We focus the analysis on transactions instead of requests** because each transaction is a set of requests, and some of them are computationally heavy. When the service gets overloaded the transactions will fail at those heavier requests. Thus, the **successful transactions is a more informative indicator of a correct service delivery**.

Table 3.2: Complete experiment Performance Metrics

Configuration	**Metric**	
	Successful Transactions (%)	*Successful Requests (%)*
No Adaptation	86.08	95.08
HPA	91.50	96.58

From Table 3.2, using HPA we get about 5% more throughput (*Successful Transactions*) than without using HPA. This was the minimum expected behavior when using HPA. Nevertheless, the throughput increase does not seem considerable. But looking at Fig. 3.13 we see that only 2 of the 20 slots (seconds 3300 to 3660 and 3900 to 4200) impose a load that cannot be handled by the default deployment. It indicates the throughput growth with HPA is a good improvement considering that only a small part of the experiment requires scaling.

The higher peaks in the workload will increase resource consumption and trigger a scale up which might take some time. This may occur primarily due to the time it takes the metrics server to update the resource consumption values, and then to the time it takes new replicas to get ready. For this workload, this behavior could not have been predicted, as the load increases abruptly. In a different scenario where the workload increases smoothly, it would be possible to proactively dispatch an adaptation.

Moreover, immediately after the slot with the highest peak, another slot with a low load comes followed by a very high one (slots 12 to 14, around seconds 3300

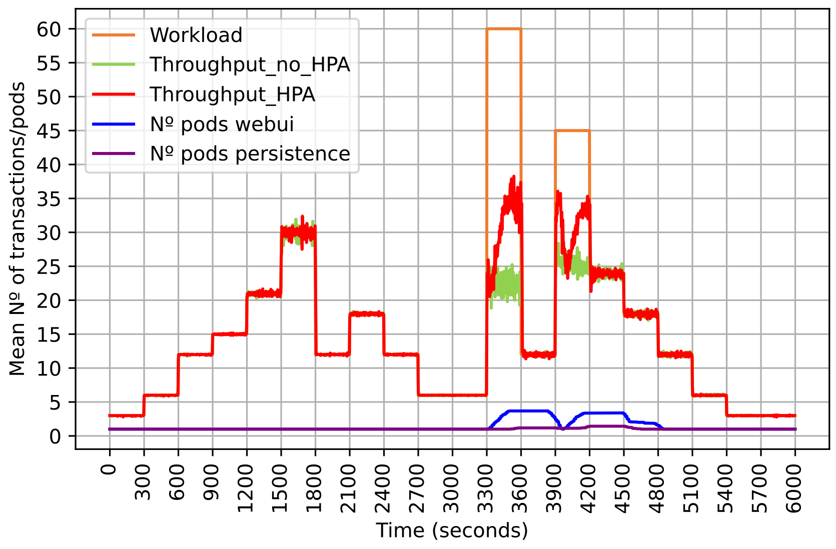


Figure 3.13: Mean throughput and mean number of pods with HPA when running the workload

to 4200). This will lead HPA to scale down after the first high peak at slot 12. As HPA has a stabilization period of 5 minutes, which is the same as the duration of workload slots, the scale down action will be triggered at the start of the second highest load peak (slot 14, second 3900 in Fig. 3.13), thereby harming performance. Also, scale-up actions will be taken later than when they should, since `metrics-server` takes 60 seconds to update the resources consumption. This delay can be seen through the number of WebUI pods which takes a while to increase at slots 12 and 14, between seconds 3300 to 3600 and 3900 to 4200. The delay to scale up will also delay scale downs due to the stabilization period of HPA. Hence, the over provisioning state will last longer as it can be seen at slot 15 (seconds 4200 to 4500).

Interestingly, the Persistence service was not scaled as many times as WebUI and did not follow its number of replicas as initially thought, like Fig. 3.14 depicts. Also, WebUI service was consistently scaled in the moments of high load, reaching its maximum number (4) of replicas.

Observing the *Successful Transactions (Txs)* on phase 1 (Table 3.3), we can see that the behavior of the system is the same with or without HPA. That is because, in phase 1, the maximum value of simultaneous users reached is 30 being the maximum tolerable 35. Thus, HPA did not take any action.

On the other hand, the results on phase 2 (Table 3.4) stand out better, meaning there is an improvement when HPA is used. Although phase 2 includes a majority of slots where HPA will not make a difference, an improvement close to 9% is achieved.

In terms of workload phases, the impact of using HPA only becomes visible in phase 2, in the slots that go over the limit tolerable by the default deployment. Hence, Table 3.5 presents the performance metrics in those slots, where an improvement close to 17% is reached in the critical load moments.

Table 3.3: Experiment Phase 1 Performance Metrics

Configuration	**Metric**	
	Successful Transactions (Txs) (%)	***Successful Requests (Reqs) (%)***
No Adaptation	99.89	99.86
HPA	99.89	99.86

Table 3.4: Experiment Phase 2 Performance Metrics

Configuration	**Metric**	
	Successful Txs (%)	***Successful Reqs (%)***
No Adaptation	75.44	90.86
HPA	84.90	93.76

Table 3.5: Experiment Slots 12 and 14 Performance Metrics

Configuration	**Metric**	
	Successful Txs (%)	***Successful Reqs (%)***
No Adaptation	52.73	87.95
HPA	69.71	92.02

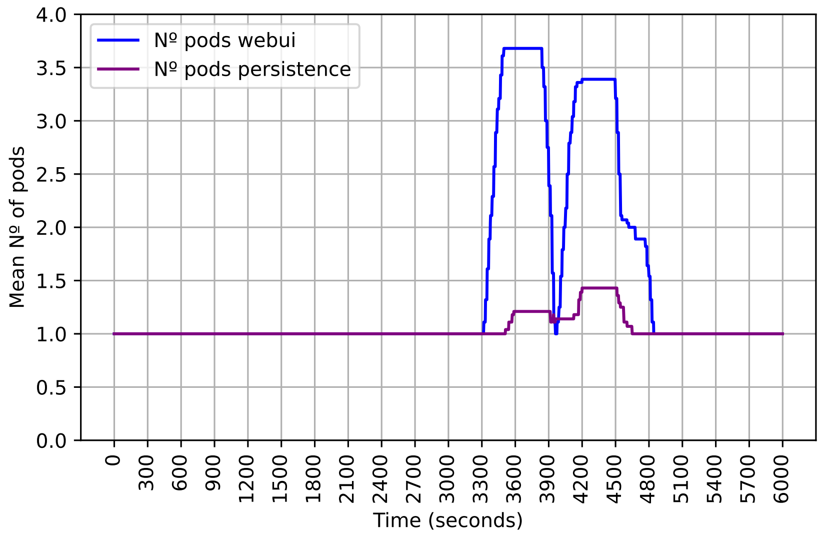


Figure 3.14: Mean number of Persistence and WebUI pods with HPA when running Workload

Looking at the throughput growth (Table 3.5) and how the services scaled (Fig. 3.14), we find interesting that the growth was not higher since there were resources left. Regarding WebUI, the highest mean number of pods was closer to 3.5 meaning that, in the most demanding period, almost half of the runs had 3 replicas while the other half had 4 replicas. Thereby, the service shouldn't need more than 4 replicas to handle the load. As for Persistence, it never needed more than 2 replicas when it could scale up to 4.

That being said, the number of each service pods handling the load was supposedly enough, but the proportion of Successful Transactions did not turn out to be as good as we thought it would. Some of the reasons that might led to these results could be scale downs in moments of high load and delayed deploys of new replicas, caused by HPA stabilization and `metrics-server` periods respectively. Also, the functioning of TeaStore may be a problem.

Based on the results, we can observe that the growth of throughput can be analyzed in various contexts but, if we consider the whole experiment, throughput had a growth of approximately 5%.

Results Considering Resource Consumption

For resource consumption, we used the ratio **R** (3.1), which measures the throughput increase in comparison to how much resources were provided. **The value obtained for the metric was** `0.94`. This result means that the proportion of added resources resulted in a lower proportion of throughput growth.

Although the metric result represents a bad value, meaning that we waste more resources to gain few on throughput, we point out that this value is an above estimate and, still, close to 1. Considering this and the problems identified in the performance metric results, we highlight that some adjustments on the `metrics-server` and HPA stabilization periods could improve the results. Thus, making the extra resources compensate for the throughput growth.

We can conclude that the proportion of throughput speedup costs almost the same or more proportion in extra resources. In this case, the proportion of throughput growth costs more 6% in extra resources using HPA. Thus, we could say this behavior is not elastic.

3.2.3 Threats to Validity

In this section, it is presented some points that may affect the validity of the results. The first is the use of a **single system under test** since we focused the assessment of HPA efficiency on a single application, TeaStore. Although it is well documented and increasingly adopted, there is space for further experiments with other microservice applications.

The second threat is the use of a **private cloud environment**. As the analysis is made upon a local Kubernetes cluster, the obtained results could be compared with the ones from a similar experiment in a public cloud environment (e.g.,***EKS*** [12]).

Other threats identified are the HPA stabilization and metrics server configurations that were used. As for the stabilization period, it was used the default value of 5 minutes that clearly did not fit the workload. Regarding the `metrics-server`, it was used an update frequency of 60 seconds, which may not be the best, as it may take longer to detect a CPU usage increase. Thus, adaptation may have been delayed and the number of successful transactions not been as high as it could.

3.2.4 Summary

To perform the experiment the % of successful transactions (set of requests that compose a complete activity in TeaStore, such as buying a product) were analyzed. Also, a formula representing a relation between the throughput gains and the increase of resources during the experiment was used. From the analysis of the results obtained it was noticed more transactions are served when HPA is used to scale application components in high load periods. It was also seen, from the formula's results, that the independent scaling of the services led to inefficient increases in production resources as this increase did not compensate for the throughput growth.

The results showed that the use of self-adaptation allows the increase of performance, throughput in this case. Also, concerning the formula used to present a relation between throughput gains and increase of resources, it did not show expected results which would be a greater throughput increase when extra resources were added. However, the experiment performed did not try more than one configuration for HPA. Since the experiment performed collected metrics from pods every 60 seconds, that may have delayed the adaptations. That could be the reason why the formula's results were not the expected as shorter metrics collection periods could and should have been tested. Besides, those shorter periods could have also resulted in a greater increase in throughput.

Kubernetes HPA is a self-adaptation enabling tool that allows scaling actions to meet demand and improve performance. However, as seen in this experiment, this is a very restrictive tool, only applicable to Kubernetes environments, and that only allows scaling pods as adaptations. For these reasons, it makes sense to use more flexible and wider applicable self-adaptation enabling tools, such as TMA.

Chapter 4

Requirements

In this chapter, the functional requirements are summarized and explained while non-functional requirements and implementation restrictions are also addressed. Respecting the functional requirements, a lot of effort was spent detailing them. Thereby, inserting them in this chapter would occupy a lot of space. For these reasons, they had to be summarized, but the detailed information can be found and read in Appendix A.

For the elicitation process of functional requirements, a use cases approach was used. Use cases are descriptions of how users will accomplish goals using the system [94]. They define how the interactions will happen, defining requests and responses both from the users and the system.

Use cases are beneficial because they help explain and define how the system should behave and identify what could go wrong in the interactions with the users. Besides, they allow establishing the cost and complexity of a system, which can then be used to negotiate which functions are built [95].

The summarized expected functional behavior of the system will be explained with the help of use case diagrams. However, and as explained before, since the descriptions of the use cases are too long, for additional and more detailed information Appendix A should be consulted.

4.1 Functional Requirements

Before diving into the enumeration and explanation of functional requirements, some Trustworthiness Monitoring & Assessment Framework (TMA) concepts must be first introduced and explained:

- **Resources:** Resources are the monitored systems from which raw data is collected. A system may have several components being monitored which translate into multiple resources. For example, a microservices application could have each of its services deployed as containers, and each of those containers is considered a resource to monitor independently. Other examples could be the monitoring of resource consumption from a machine containing multiple Virtual Machines (VMs), where each one of them could be considered a resource and monitored independently, or we could consider the guest machine as a single resource to monitor;

- **Descriptions:** Descriptions are used to identify and describe raw data collected by Probes. Also, Metrics which are leaf attributes on a Quality Model tree use these descriptions to describe the data they are related to. For example, CPU consumption could be a leaf metric that would be associated with a Description that would refer to % CPU data collected from Probes;
- **Probes:** Probes are the components in charge of gathering raw information from systems states. They collect raw data from monitored resources which are then used to compute the metrics values;
- **Metrics:** Metrics are what compose the Quality Models. They represent the properties of a system and thereby they are what is monitored. They can be of 2 types: *Metric* and *Leaf Metric*. The first has its values calculated based on weights applied to its child metrics. *Leaf Metrics* have their values calculated by processing raw data collected by probes;
- **Quality Models:** Quality Models define hierarchical trees of metrics. To those trees are applied configuration profiles to create weighted metrics trees that are then used to calculate scores to reflect systems states;
- **Configuration Profiles:** These components define weights to apply for each metric of a Quality Model tree;
- **Actuators:** Actuators are responsible for executing adaptations on a system. Thereby they are what assures adaptation plans are executed by receiving orders from TMA's Execute component;
- **Actions:** Actions define requests associated with actuators that are invoked to trigger adaptations on Resources. Actions may have configurations which are the parameters that customize an Action to be performed in different manners. This means that there an Action can be executed multiple times but with different values for its configurations;
- **Adaptation Rules:** Adaptation Rules are conditions that when violated generate the creation of plans. TMA's Planning is the component responsible for verifying these rules;
- **Plans:** Plans are created by adaptation rules and they represent an ordered execution of actions. These Plans are orchestrated by TMA's Execute component;
- **Logs:** Logs are important events on TMA's platform that need to be saved to provide an alternative view on its management.

The requirements of the dashboard to be created can be grouped and then categorized into 4 topics using TMA's related concepts:

- **Probe context:** groups related functionalities to the systems monitoring process. Therefore, the TMA's concepts *Resources*, *Descriptions* and *Probes* are here tied together. *Resources* represent systems or parts of them, *Descriptions* detailed information on what is collected from monitored systems, and *Probes* the units that are responsible for collecting the descriptions from systems. As systems are represented in this category, functionalities related to the visualization of metrics and performed adaptations are included;

- **Quality Model context:** groups related functionalities to the management of the TMA's concept *Quality Models*. As this concept is coupled with *configuration profiles* and *metrics* concepts to represent weighted trees of metrics, all of these concepts had to be included. *Configuration profiles* define the weights applied on the tree while *Metrics* are the base and core unit of those trees;

- **Adaptation context:** reunites a set of functions focused on adaptations. Hence, the *tma*'s concepts of *Actions*, *Actuators*, *Configurations* and *Adaptation rules* are presented here. *Actions* define operations to execute on systems and they are constituted by *Actuators*, which are responsible for executing the actions on the systems, and *Configurations*, which define parameters used to personalize the operations performed by actuators. *Adaptation Rules* define conditions that may trigger adaptations;

- **Decision support extras context:** although the extras may sound like requirements that could not be present in the product, they represent additional functionalities to ease the management of systems and help in the decision process.

In the following sections, each of the 4 categories just presented will have their set of requirements summarized. As pointed out before, the details of each use case can be found in Appendix A of this document.

4.1.1 Probe context

For this category the requirements, presented by Fig. 4.1, include functionalities for reading, creating, updating and deleting *Resources*, *Descriptions* and *Probes*.

Resources are the monitored systems and, for each of them, there is a configuration profile applied to control the current state. So, this association had to exist as functionality so that it becomes possible to know what quality model, with which weighting, has to be applied when managing a certain resource. Besides, the configuration profile for a resource is not fixed because later there may exist a need for updating it. Thus, the possibility of changing it was also added as a requirement.

As this context defines probing-related functions, functionalities for visualizing the values of metrics from a system are located in this context. Basically, through charts, it is possible to see the metrics values from a resource over time. Optionally, alongside the metrics, an indication of adaptation plans that took place can also be viewed. These charts can then be exported for any purposes the user intends, such as articles and presentations. Also, as the product is a dashboard, a requirement for exporting those chart configurations to a file was added. Thus, that file can later be used to add charts to the homepage to facilitate the monitoring activity on systems. Besides, those files can be manually altered to generate new charts that can be added to the homepage.

There is also an option for simulating metrics values from a resource. When this option is chosen, users perform a simulation where they are allowed to change some of the weights of the metrics tree associated with the resource and see what would be the new values for the metrics, if those weights were applied.

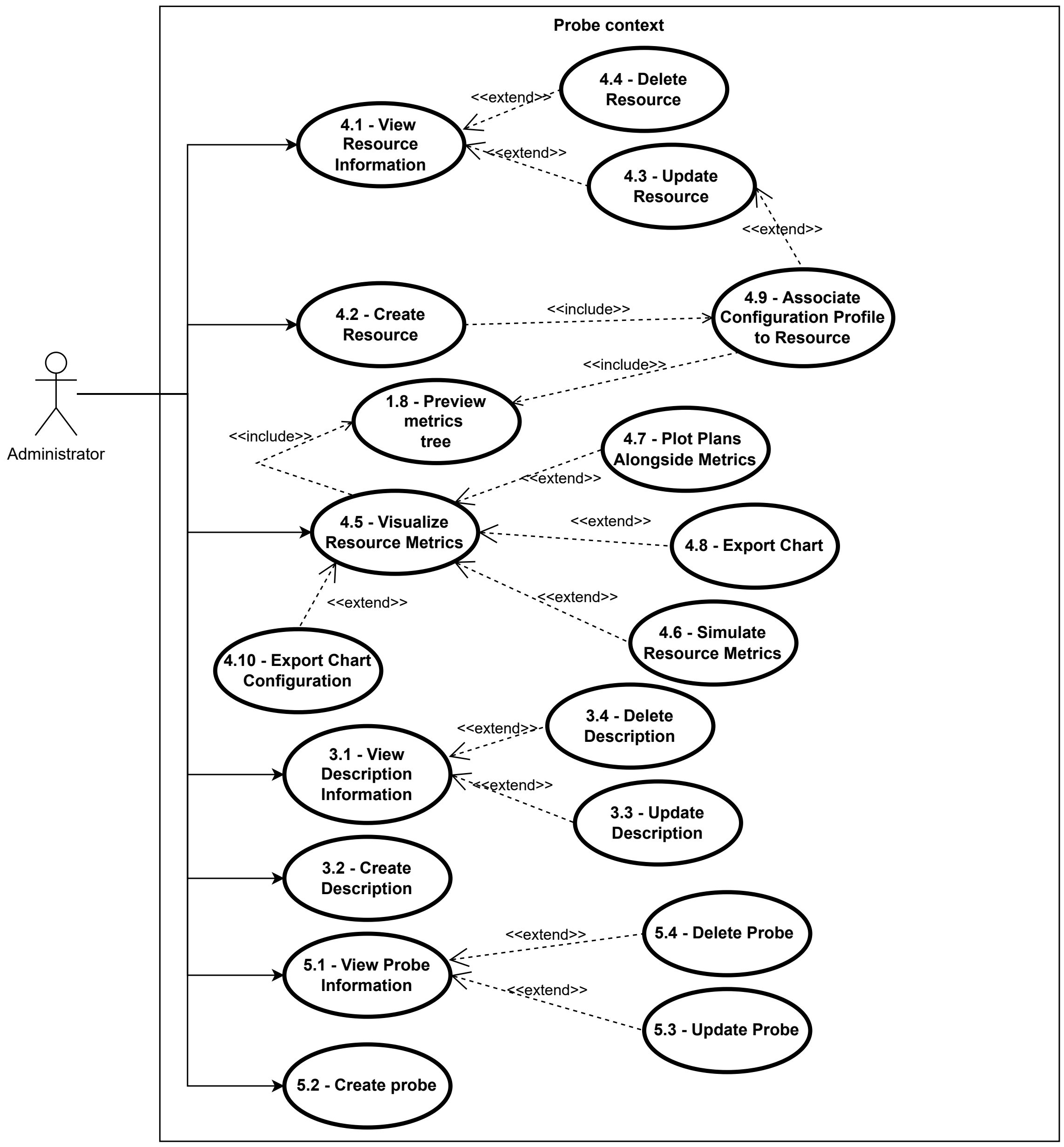

Probe context
Administrator
4.1 - View Resource Information
4.4 - Delete Resource
4.3 - Update Resource
4.2 - Create Resource
4.9 - Associate Configuration Profile to Resource
1.8 - Preview metrics tree
4.7 - Plot Plans Alongside Metrics
4.5 - Visualize Resource Metrics
4.8 - Export Chart
4.6 - Simulate Resource Metrics
4.10 - Export Chart Configuration
3.1 - View Description Information
3.4 - Delete Description
3.3 - Update Description
3.2 - Create Description
5.1 - View Probe Information
5.4 - Delete Probe
5.3 - Update Probe
5.2 - Create probe
<<extend>>
<<include>>

Figure 4.1: Probing Related Use Cases Diagram.

4.1.2 Quality Model context

The requirements elicited within this category, presented by Fig. 4.2, include functionalities for reading, creating, updating and deleting *Quality Models*, *Configuration Profiles* and *Metrics*.

In the creation of *Metrics* it may be possible to create a leaf metric and, therefore, associating it to a description had to be added as a requirement because those metrics use the data collected from systems to have their value calculated. As there are hierarchies of metrics, associations between parent and child metrics also had to be a requirement.

Quality Models are the representations of metrics trees so, there is a requirement for the association between a quality model and the metric it wants to use as a root node. However, for the same quality model, there may exist different weightings to be applied. Thus, for a given quality model multiple configuration profiles can be viewed or created.

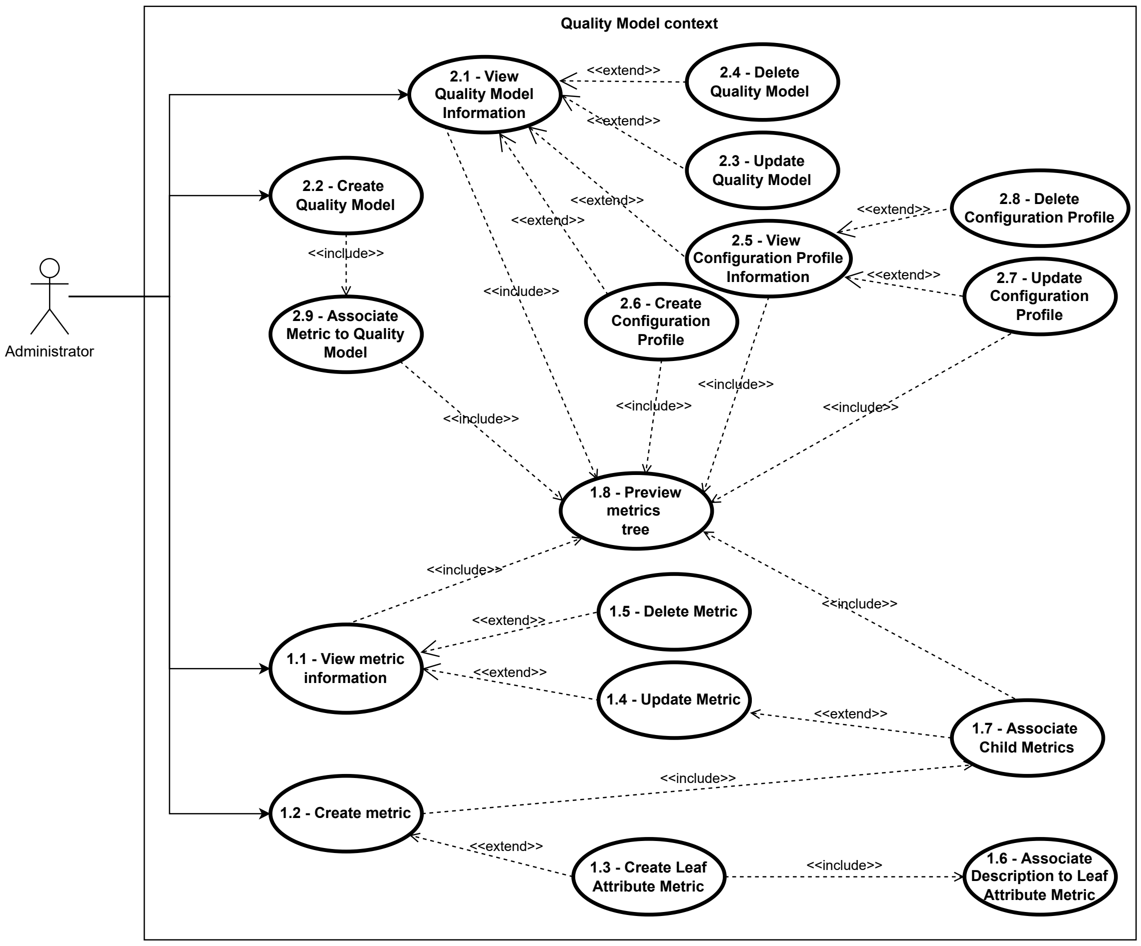

Quality Model context
Administrator
2.1 - View Quality Model Information
2.2 - Create Quality Model
2.3 - Update Quality Model
2.4 - Delete Quality Model
2.5 - View Configuration Profile Information
2.6 - Create Configuration Profile
2.7 - Update Configuration Profile
2.8 - Delete Configuration Profile
2.9 - Associate Metric to Quality Model
1.8 - Preview metrics tree
1.1 - View metric information
1.2 - Create metric
1.3 - Create Leaf Attribute Metric
1.4 - Update Metric
1.5 - Delete Metric
1.6 - Associate Description to Leaf Attribute Metric
1.7 - Associate Child Metrics
<<extend>>
<<include>>

Figure 4.2: Quality Models Related Use Cases Diagram.

The update functionalities are essentially metadata editing, except for configuration profiles where updating the weights of a metrics tree is allowed. Updating metrics is also not only about metadata because parent and child associations can be performed if the metric being updated is not a leaf attribute and is not associated with a quality model that has been used.

Finally, there is a use case that is many times incorporated in others which is the "Preview metrics tree". This is one of the most important features of the product to develop as it assists in many other functionalities. It provides a graphical view for the user when performing interactions related to quality model concepts.

4.1.3 Adaptation context

Once again the requirements, presented by Fig. 4.3, include functionalities for reading, creating, updating and deleting, but this time considering *Actions*, *Actuators* and *Adaptation Rules*.

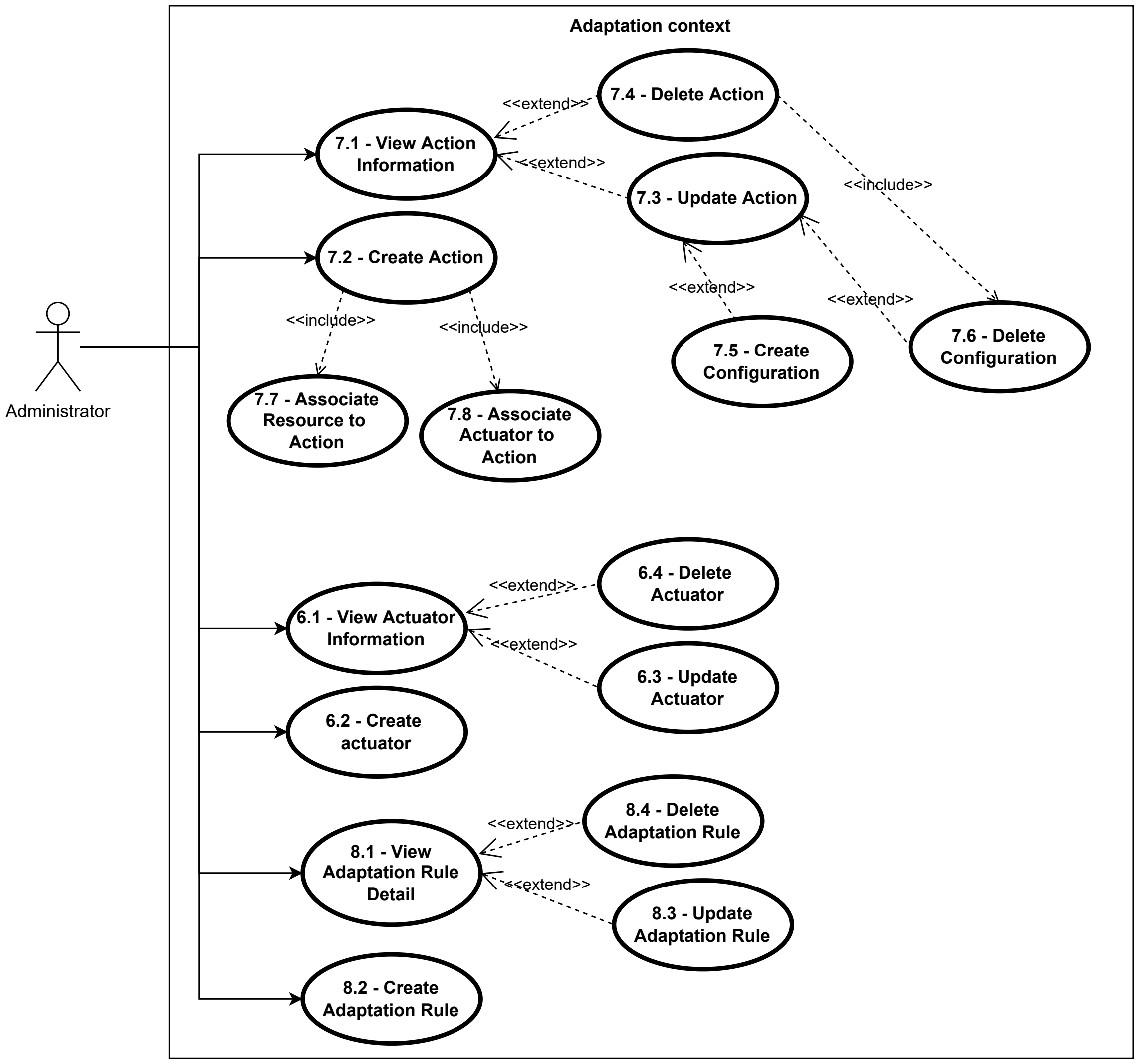

Adaptation context
Administrator
7.1 - View Action Information
7.2 - Create Action
7.3 - Update Action
7.4 - Delete Action
7.5 - Create Configuration
7.6 - Delete Configuration
7.7 - Associate Resource to Action
7.8 - Associate Actuator to Action
6.1 - View Actuator Information
6.2 - Create actuator
6.3 - Update Actuator
6.4 - Delete Actuator
8.1 - View Adaptation Rule Detail
8.2 - Create Adaptation Rule
8.3 - Update Adaptation Rule
8.4 - Delete Adaptation Rule
<<extend>>
<<include>>

Figure 4.3: Actuating Related Use Cases Diagram.

An Action is always associated with a monitored Resource and an Actuator that performs adaptations. So, those associations had to be elicited on the requirements. Also, an Action has *Configurations* which represent customized parameters accepted by an Actuator to perform operations. Thus, the product should allow to create and delete, if necessary, Action's configurations.

Moreover, what triggers the creation of adaptation plans, which are formed by a set of actions, are the *Adaptation Rules*. So, the system should allow users to manage those conditions that trigger adaptations, by creating, editing, or deleting them.

4.1.4 Decision support extras context

The requirements drawn from this category are represented by Fig. 4.4. It essentially contains a set of features the system must have to improve the systems management activity. Therefore, the system to develop should allow listing of performed adaptation plans and consulting details of them, such as actions performed, with which configurations, and using which actuator.

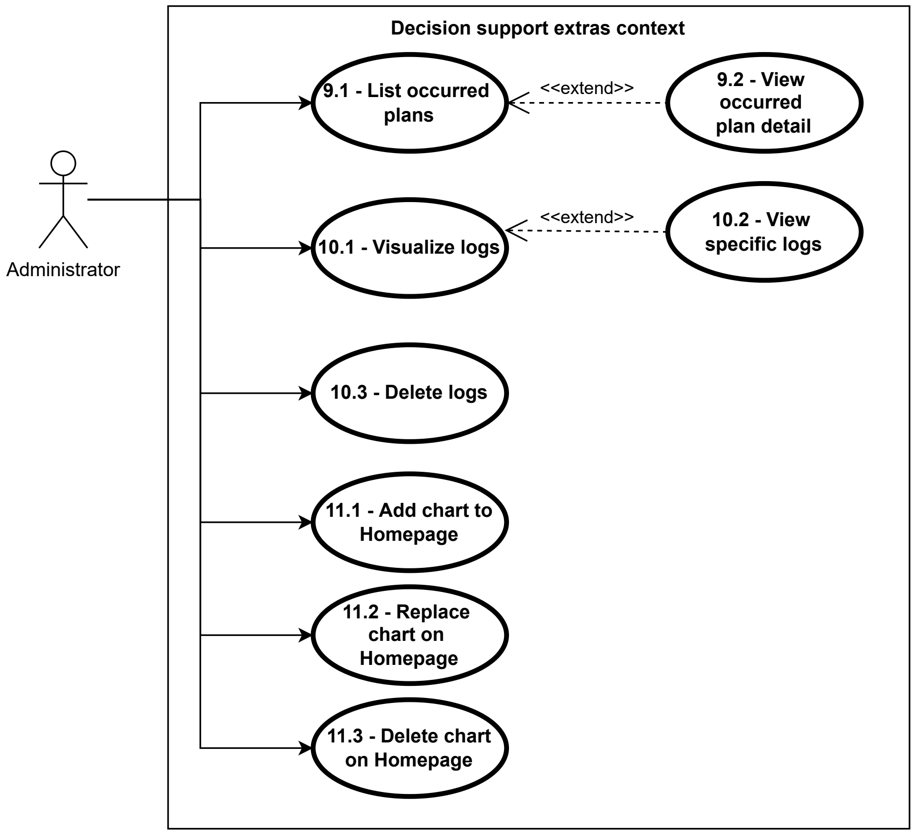

Decision support extras context
Administrator
9.1 - List occurred plans
<<extend>>
9.2 - View occurred plan detail
10.1 - Visualize logs
<<extend>>
10.2 - View specific logs
10.3 - Delete logs
11.1 - Add chart to Homepage
11.2 - Replace chart on Homepage
11.3 - Delete chart on Homepage

Figure 4.4: Decision Support Extras Use Cases Diagram.

Furthermore, the system should have a logs mechanism that will provide users with information from overall system behavior, giving insights e.g., on quality models changes and adaptation plans. The system should also provide filtering options to allow visualizing logs of a specific kind instead of all of them. And in case the database starts increasing due to these logs, it should be possible to delete some that are not considered to be needed anymore.

Finally, as the main goal is to develop a dashboard, requirements for managing (add, replace, and delete) charts on a homepage were added. These features have the purpose of providing an easy and fast accessibility point for users to monitor their systems.

4.1.5 Implementation status

The use case requirements presented in the previous sections are all the requirements elicited, however, as foreseen, not all of them would and could have been implemented during this dissertation. That is why priorities were assigned to use cases based on their influence on the success of the main goal of the dissertation, the creation of the dashboard. Thus, Table 4.1 presents the use cases ids, names, priorities, and implementation statuses.

Table 4.1: Use cases implementation statuses.

Id	Name	Priority	Implemented
1.1	View Metric information	Must	Yes
1.2	Create Metric	Must	Yes
1.3	Create Leaf Attribute Metric	Must	Yes
1.4	Update Metric	Won't	No
1.5	Delete Metric	Won't	No
1.6	Associate Description to Leaf Attribute Metric	Must	Yes
1.7	Associate Child Metrics	Must	Yes
1.8	Preview metrics tree	Must	Yes
2.1	View Quality Model Information	Must	Yes
2.2	Create Quality Model	Must	Yes
2.3	Update Quality Model	Won't	No
2.4	Delete Quality Model	Won't	No
2.5	View Configuration Profile Information	Must	Yes
2.6	Create Configuration Profile	Must	Yes
2.7	Update Configuration Profile	Won't	No
2.8	Delete Configuration Profile	Won't	No
2.9	Associate Metric to Quality Model	Must	Yes
3.1	View Description Information	Could	No
3.2	Create Description	Could	No
3.3	Update Description	Won't	No
3.4	Delete Description	Won't	No
4.1	View Resource information	Could	No
4.2	Create Resource	Could	No
4.3	Update Resource	Won't	No
4.4	Delete Resource	Won't	No
4.5	Visualize Resource Metrics	Must	Yes
4.6	Simulate Resource Metrics	Must	Yes
4.7	Plot Plans Alongside Metrics	Must	Yes
4.8	Export Chart	Must	Yes
4.9	Associate Configuration Profile to Resource	Could	No
4.10	Export Chart Configuration	Must	Yes
5.1	View Probe Information	Could	No
5.2	Create probe	Could	No
5.3	Update Probe	Won't	No

Id	**Name**	**Priority**	**Implemented**
5.4	Delete probe	Won't	No
6.1	View Actuator Information	Could	No
6.2	Create actuator	Could	No
6.3	Update Actuator	Won't	No
6.4	Delete actuator	Won't	No
7.1	View Action information	Could	No
7.2	Create Action	Could	No
7.3	Update Action	Won't	No
7.4	Delete Action	Won't	No
7.5	Create Configuration	Could	No
7.6	Delete Configuration	Could	No
7.7	Associate Resource to Action	Could	No
7.8	Associate Actuator to Action	Could	No
8.1	View Adaptation Rule Detail	Must	Yes
8.2	Create Adaptation Rule	Must	Yes
8.3	Update Adaptation Rule	Won't	No
8.4	Delete Adaptation Rule	Must	Yes
9.1	List occurred plans	Should	No
9.2	View occurred plan detail	Should	No
10.1	Visualize logs	Should	No
10.2	View specific logs	Should	No
10.3	Delete logs	Won't	No
11.1	Add chart to Homepage	Must	Yes
11.2	Replace chart on Homepage	Must	Yes
11.3	Delete chart on Homepage	Must	Yes

Only and all the requirements set with the "Must" priority were implemented.

4.2 Non-functional requirements and restrictions

Design and implementation restrictions are those constraints that are imposed on the design solution and thereby must be fulfilled when designing and implementing the product. They are typically imposed by the customer, by the development organization, or by external regulations. They may be imposed on the hardware, software, data, operational procedures, interfaces, or any other part of the system [96].

For the product to be developed, a dashboard, the following restrictions were identified:

- The product should be developed as a web page;
- It must be accessible over the Network;

- It must be deployable through Kubernetes;
- No direct communication with TMA components. Requests must pass through the TMA-API component.

Non-functional requirements, also known as quality attributes, represent properties a system should have [97]. This type of requirement influences design/implementation decisions. The ones identified in this scope are:

- **Usability -** The focus of this requirement will not be on tasks of the dashboard that are direct and simple (e.g., creating a leaf attribute metric, and creating a quality model). Instead, the focus will be on tasks requiring some effort and attention from users, such as simulating metrics, plotting metrics, and management of the Homepage plots. The focus is on these tasks because they represent features that most contribute to the product (dashboard) value. Also, a quantitative and qualitative assessment will be performed. Therefore, for technological users that comprehend the goals of TMA and know how it works at a high level, in a first interaction with the dashboard the following requirements should be fulfilled:
  - Every user should complete 80 % of the planned tasks;
  - Tasks completed should be done with the maximum of 3 errors;
  - Tasks do not take longer to complete than the expected maximum duration;
  - 100 % of the users can correctly identify 6 out of the 8 parameters that constitute the exported chart configuration files and their types of data;
  - 75 % of the users can tell when simulations can be performed;
  - 75 % of the users can tell the period and what is going to be simulated when they are at the simulation window;
  - An average of 75 % positive answers is received from users on a qualitative form's questions.
- **Performance -** Since it is a React application, the web app will have its processing at the client side because the product will be a Single Page Application (SPA). Also, as the frontend works along with the TMA's API, these two entities have their performance linked. However, the product is meant to be used by users with technological knowledge, more precisely, knowledge concerning TMA. That means the product is not intended to have many users, but instead a small group of them. That being said, both the frontend and the Application Programming Interface (API) need to fulfill requirements in different aspects:
  - Having the dashboard's homepage full with live data plots, CPU consumption should not be higher than 1 virtual CPU and RAM consumption should not be higher than 250 MB;
  - The web page should not take longer than 3 seconds to be loaded since it is the majority loading time users wait before leaving [98];

  - A single kubernetes pod of the API, consuming at maximum 1 virtual CPU, should be capable of handling, at minimum, 100 requests per second.

- **Maintainability -** There are plans for using this tool, so maintainability must be kept in mind when developing the product as it is intended to be changed, enhanced, and maybe restructured over time. Additionally, errors may arise and the task of fixing them should be eased. Thus, facilitating the process of future changes to the product must be assured.

  Thus, to ensure maintainability, when developing, high cohesion and loose coupling approaches should be followed, besides naming and other technologies conventions. Clear comments and documentation should also be created so that it becomes easier to track code to be improved or corrected.

Chapter 5

Architecture and Implementation

In this chapter, architecture and implementation are presented, described, and explained. First, the architecture is addressed beginning with a contextual view. Then, other and more detailed perspectives are shown to complement the architecture's description. Finally, implementation insights are given starting with Trustworthiness Monitoring & Assessment Framework (TMA)'s maintenance, followed by the components changed and implemented during the dissertation.

5.1 Architecture

This section presents the proposed architecture for the product, starting with high-level diagrams and ending with more detailed ones. To create and design those diagrams the ***C4 model*** [99] was used. It stands for context, containers, components, and code. This model is composed of hierarchical diagrams that can be used to describe software architecture at different zoom levels [100].

To fully understand the architecture that is going to be proposed in this section, the context in which the solution is inserted must be understood first. Fig. 5.1 presents that context.

From a high-level perspective, there are 2 main types of interactions that take place in the environment involving TMA. The first is the interactions a user, **TMA Administrator**, has with TMA for configuring the platform and performing systems management activities. The other type of interaction is the monitoring and adaptation activities performed autonomously by TMA to adapt monitored systems.

Now that the context has been presented, it is time to dive deeper into details of how exactly the solution to be developed fits in the environment since it will be part of TMA's platform. To understand that, the architectural diagram presented by Fig. 5.2 describes in more detail how TMA is composed as well as the managed system environment.

TMA is fundamentally composed of a MAPE-K control loop, explained in sec-

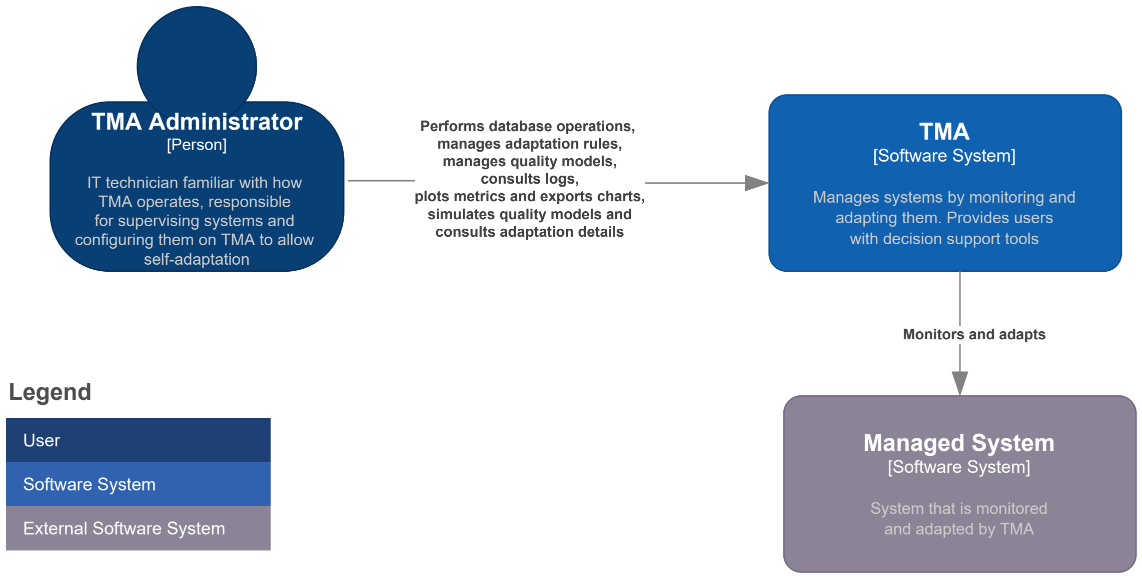

TMA Administrator
[Person]
IT technician familiar with how TMA operates, responsible for supervising systems and configuring them on TMA to allow self-adaptation
Performs database operations, manages adaptation rules, manages quality models, consults logs, plots metrics and exports charts, simulates quality models and consults adaptation details
TMA
[Software System]
Manages systems by monitoring and adapting them. Provides users with decision support tools
Monitors and adapts
Managed System
[Software System]
System that is monitored and adapted by TMA
Legend
User
Software System
External Software System

Figure 5.1: Context architectural diagram.

tion 3.1.1, and other elements that complement the platform and that loop. Also, it implements a microservices architecture, which means each component that composes TMA can leverage containerization. Therefore, those components are implemented as containers to take advantage of the benefits the technology and microservices offer.

In the control loop there are the **Monitor**, **Analyze**, **Planning**, **Execute** and **Knowledge** elements. As described before, *Monitor* is responsible for receiving data from systems and persisting it at the *Knowledge*. Those data are then processed using quality models and saved persistently by the *Analyze*. *Planning* reads the processed data, which represents systems states, and compares it against a set of rules. In case of violations of those rules, plans are created and their details saved on the *Knowledge*. Later, *Execute* will read the details of the plans and orchestrate adaptations. *Knowledge* is just the repository for the configurations of *TMA* and other data that is needed for the components to perform monitoring and adaptations of systems.

Outside the control loop, but still a part of *TMA system*, there are the **TMA-API** and **Dashboard** elements. *TMA-API* is a classic Application Programming Interface (API) that provides functionalities through endpoints which can be invoked by the *Dashboard* to provide users with functionalities. Basically, *TMA-API* is responsible for creating, reading, updating and deleting data on the *Knowledge* database, and applying any logic to serve requests.

Externally to the TMA system, but integrated with, there are **Probes** and **Actuators**. *Probes* are components in charge of collecting data from managed systems and sending it to TMA's *Monitor*. *Actuators* are agents that execute actions to perform adaptations on managed systems. They expose endpoints which are invoked by TMA's *Execute* with provided configurations which define what and how the actuator will act.

Next, and observing Fig. 5.3, more details on the architecture of the *Dashboard*,

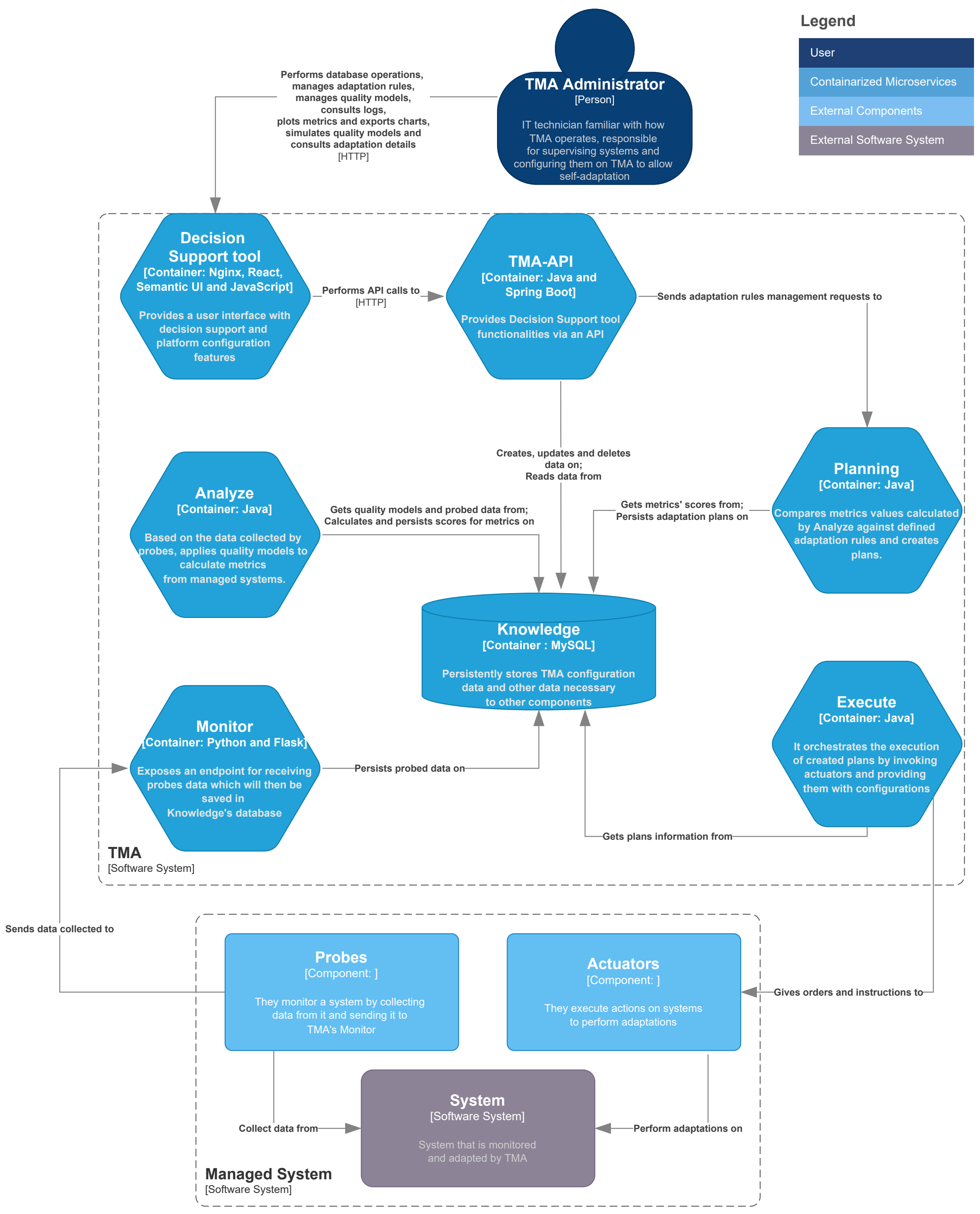

Legend
User
Containarized Microservices
External Components
External Software System
TMA Administrator
[Person]
IT technician familiar with how TMA operates, responsible for supervising systems and configuring them on TMA to allow self-adaptation
Performs database operations, manages adaptation rules, manages quality models, consults logs, plots metrics and exports charts, simulates quality models and consults adaptation details
[HTTP]
Decision Support tool
[Container: Nginx, React, Semantic UI and JavaScript]
Provides a user interface with decision support and platform configuration features
Performs API calls to
[HTTP]
TMA-API
[Container: Java and Spring Boot]
Provides Decision Support tool functionalities via an API
Sends adaptation rules management requests to
Creates, updates and deletes data on; Reads data from
Planning
[Container: Java]
Compares metrics values calculated by Analyze against defined adaptation rules and creates plans.
Gets metrics' scores from; Persists adaptation plans on
Analyze
[Container: Java]
Based on the data collected by probes, applies quality models to calculate metrics from managed systems.
Gets quality models and probed data from; Calculates and persists scores for metrics on
Knowledge
[Container : MySQL]
Persistently stores TMA configuration data and other data necessary to other components
Execute
[Container: Java]
It orchestrates the execution of created plans by invoking actuators and providing them with configurations
Monitor
[Container: Python and Flask]
Exposes an endpoint for receiving probes data which will then be saved in Knowledge's database
Persists probed data on
Gets plans information from
TMA
[Software System]
Sends data collected to
Probes
[Component:]
They monitor a system by collecting data from it and sending it to TMA's Monitor
Actuators
[Component:]
They execute actions on systems to perform adaptations
Gives orders and instructions to
System
[Software System]
System that is monitored and adapted by TMA
Collect data from
Perform adaptations on
Managed System
[Software System]

Figure 5.2: TMA and managed system architecture.

TMA's API, and *TMA's Planning* components are presented. An important note regarding this figure is that it presents the architecture designed to support all the requirements elicited. However, due to time restrictions and prioritization of functional requirements, the TMA's API components "*Log Controller*" and *"Plan Controller"* were not implemented.

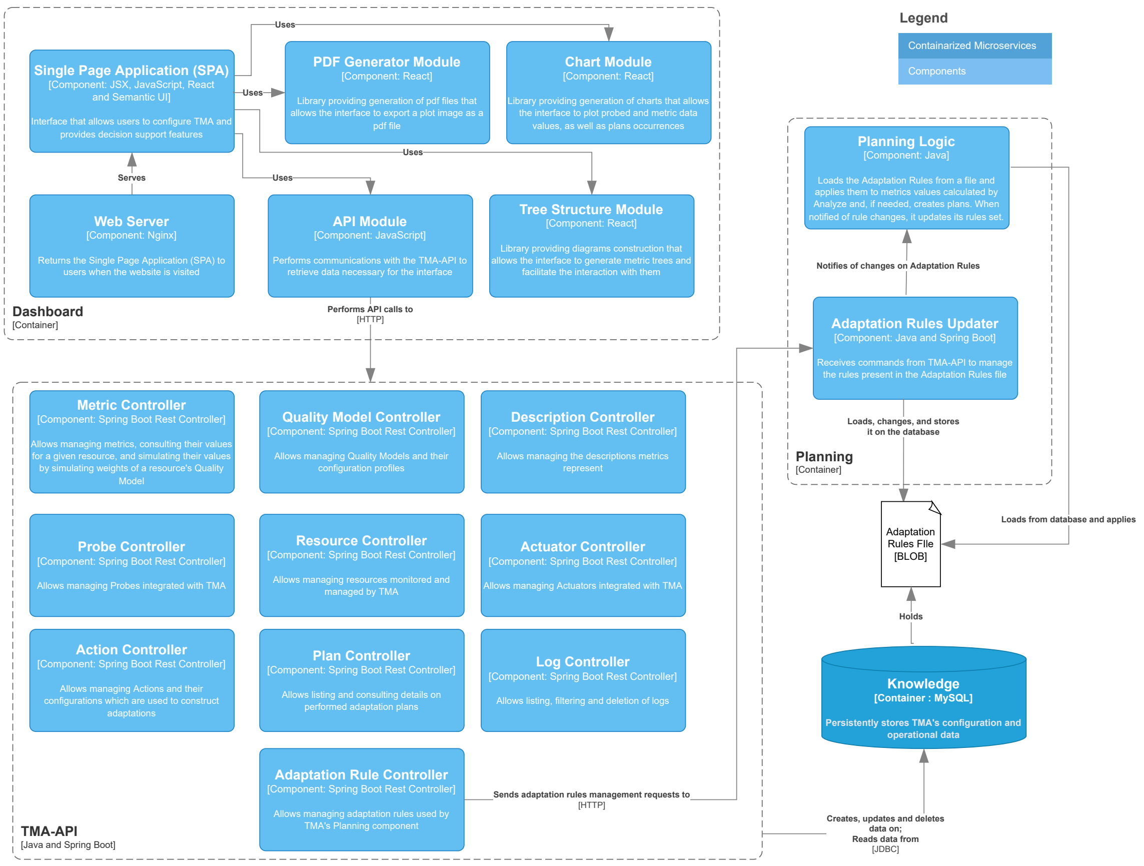

Legend
Containarized Microservices
Components
Uses
Single Page Application (SPA)
[Component: JSX, JavaScript, React and Semantic UI]
Interface that allows users to configure TMA and provides decision support features
Uses
PDF Generator Module
[Component: React]
Library providing generation of pdf files that allows the interface to export a plot image as a pdf file
Chart Module
[Component: React]
Library providing generation of charts that allows the interface to plot probed and metric data values, as well as plans occurrences
Uses
Serves
Uses
Web Server
[Component: Nginx]
Returns the Single Page Application (SPA) to users when the website is visited
API Module
[Component: JavaScript]
Performs communications with the TMA-API to retrieve data necessary for the interface
Tree Structure Module
[Component: React]
Library providing diagrams construction that allows the interface to generate metric trees and facilitate the interaction with them
Dashboard
[Container]
Performs API calls to
[HTTP]
Planning Logic
[Component: Java]
Loads the Adaptation Rules from a file and applies them to metrics values calculated by Analyze and, if needed, creates plans. When notified of rule changes, it updates its rules set.
Notifies of changes on Adaptation Rules
Adaptation Rules Updater
[Component: Java and Spring Boot]
Receives commands from TMA-API to manage the rules present in the Adaptation Rules file
Loads, changes, and stores it on the database
Planning
[Container]
Metric Controller
[Component: Spring Boot Rest Controller]
Allows managing metrics, consulting their values for a given resource, and simulating their values by simulating weights of a resource's Quality Model
Quality Model Controller
[Component: Spring Boot Rest Controller]
Allows managing Quality Models and their configuration profiles
Description Controller
[Component: Spring Boot Rest Controller]
Allows managing the descriptions metrics represent
Probe Controller
[Component: Spring Boot Rest Controller]
Allows managing Probes integrated with TMA
Resource Controller
[Component: Spring Boot Rest Controller]
Allows managing resources monitored and managed by TMA
Actuator Controller
[Component: Spring Boot Rest Controller]
Allows managing Actuators integrated with TMA
Adaptation Rules FIle
[BLOB]
Loads from database and applies
Holds
Action Controller
[Component: Spring Boot Rest Controller]
Allows managing Actions and their configurations which are used to construct adaptations
Plan Controller
[Component: Spring Boot Rest Controller]
Allows listing and consulting details on performed adaptation plans
Log Controller
[Component: Spring Boot Rest Controller]
Allows listing, filtering and deletion of logs
Knowledge
[Container : MySQL]
Persistently stores TMA's configuration and operational data
Adaptation Rule Controller
[Component: Spring Boot Rest Controller]
Allows managing adaptation rules used by TMA's Planning component
Sends adaptation rules management requests to
[HTTP]
TMA-API
[Java and Spring Boot]
Creates, updates and deletes data on;
Reads data from
[JDBC]

Figure 5.3: Dashboard, TMA's API and Planning architecture.

Respecting the *Dashboard*, which is the main goal of this dissertation, it is a **Single Page Application (SPA)** provided to users through an Nginx **Web Server**. The SPA interface is written in JavaScript Syntax Extension (JSX), implemented using React components and Semantic UI for styling as defined in Section 2.4.8. To provide the elicited functionalities, this interface makes use of other elements: **PDF Generator Module**, **Chart Module**,**API Module**, and **Tree Structure Module**. *PDF Generator Module* is a library that provides functionalities to the generation of Portable Document Format (PDF) files, which will be used by the interface to export a plot as an image in a PDF file. *Chart Module* is also a library, which will allow the interface to generate charts with probed and metric data values, as well as plan occurrences. The *Tree Structure Module* is a library that allows the interaction with and creation of metrics trees to be presented on the interface. Finally, the *API Module* consists of JavaScript code responsible for dealing with communications with TMA's API.

Using the *API Module*, the SPA performs requests to different controllers that compose *TMA's API*, where each controller holds a set of related endpoints. The information retrieved from these endpoints allows the SPA to provide features to

TMA Administrators through a User Interface (UI). Those features consist of managing TMA database for configuring the platform, and performing management activities such as: consulting logs; visualizing and exporting charts; managing and simulating quality models; consulting adaptation plans details; managing adaptation rules.

Considering adaptation rules, there is a file saved as a Binary Large Object (BLOB) in the *Knowledge* database. When changes to those rules are performed at the SPA, requests are sent to *TMA-API*. However, they will not be served there, but instead by another API server that is present in *TMA's Planning*. This API is represented by the element **Adaptation Rules Updater** which will then update the adaptation rules file and notify **Planning Logic** element of those changes. *Planning Logic* initially loads the rules from the file present in the database to decide on creating or not adaptation plans. As notifications of changes in the rules arrive, this element will update its current adaptation rules set.

5.2 Implementation

At a first glance, this section will provide adjustments that were made on TMA due to maintenance purposes. These adjustments could have their origins in found issues on the platform or in the necessity of features for the product developed.

Then, details drawn from requirements implementation are presented.

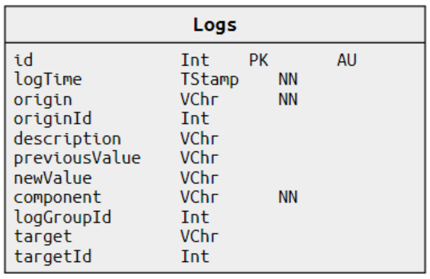
5.2.1 TMA Maintenance

Before beginning to develop the product of this dissertation, TMA was experimented to further understand its concepts, operations, and what it was missing for supporting the new UI.

Minding one of the features the product should have, **a logging system**, a new entity has been added to the TMA database to support it. The goal of having a logging system is to give TMA administrator a global insight on what is happening on the platform and help him trace related actions on a specific managed element or errors if that becomes necessary. This is achieved by creating a centralized point of access for consulting all the logs from the platform. Thereby, this is a feature that supports the management of systems and TMA's platform.

This new entity is presented on Fig. 5.4 and the data meaning of its elements is described next:

- **id:** It is the primary key, a unique identifier of each log, and is automatically incremented;
- **logTime:** It is the time the log happens;

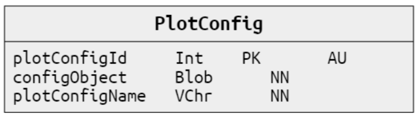


Logs				
id	Int	PK		AU
logTime	TStamp		NN	
origin	VChr		NN	
originId	Int			
description	VChr			
previousValue	VChr			
newValue	VChr			
component	VChr		NN	
logGroupId	Int			
target	VChr			
targetId	Int			

Figure 5.4: Logs database entity.

- **origin:** It is the subject that dispatches the log. For example, it can be a Plan or a Quality Model;
- **originId:** It is an identifier related to the subject that dispatched the log. For example, in the case of a Plan, it is the planID;
- **description:** It is a text that clarifies the log, something readable that makes it easy to understand what the log is about. For example, in the Plan's case, something like "Executing Plan activated by the rule MyRule";
- **previousValue:** This is an identifier of the old value related to a context. For example, in Quality Models' context, it is the previous value of some weight. In the case of a Plan, it could be the previous number of instances of a resource;
- **newValue:** This is an identifier of the new value related to a context. For example, in Quality Models' context, it is the new value of some weight. In the case of a Plan, it could be the new number of instances of a resource;
- **component:** It is an identifier of the component of the TMA architecture that is logging e.g., Analyze, Planning, Executer, or Monitor;
- **logGroupId:** It is an identifier of a group of logs that had the same purpose through a number identifier that is automatically assigned and incremented. For example, when changing the weights of 2 nodes on the Quality Model two logs are created and they will have this parameter with the same number identifier;
- **target:** This is a name identifying the target of some action in a context. For example, in the context of changing a Resource's instance number, the target identifies the name of the Resource;
- **targetId:** This is a number that identifies the target of some action in a context. For example, in the context of changing a Resource's instance number, targetId is the corresponding Id in the Resource database table.

Also, concerning the features related to the management of homepage charts, another entity was added to the TMA database. This new entity contains information necessary to generate a chart at the homepage of the dashboard. The new database table is presented on Fig. 5.5 and the data meaning of its elements is described next:



PlotConfig

plotConfigId	Int	PK	AU
configObject	Blob	NN	
plotConfigName	VChr	NN	

Figure 5.5: PlotConfig database entity.

- **plotConfigId**: It is the primary key, a unique identifier of each plotConfig, and is automatically incremented;
- **configObject**: It is a JavaScript object containing several configuration parameters necessary to generate a chart at the dashboard;
- **plotConfigName**: It is a name associated to the plot configuration which will be presented in the dashboard along with the generated chart.

Additionally, there was a known **TMA's issue related with plans**. An adaptation plan with more than one action would only execute the first one. Although the platform supported several actions at the database level, it did not have support when executing the adaptation plan. In practice, different plans with a single action would have to be created to perform the desired operations instead of a single one with multiple actions. Thereby, it was added support for more than one action per adaptation plan by rewriting the *Planning* and *Execute* components code.

5.2.2 Requirement implementation

This section presents implementation details on the goal product of this dissertation, the Dashboard. Also, creating this product affects how other components operate. Therefore, details on changes made to TMA's API, Analyze, and Planning components are also presented next.

Dashboard

To comply with the defined implementation restrictions on Section 4.2, the dashboard has a Dockerfile that generates an image containing an Nginx server that will reply to incoming requests by returning the code that composes the SPA. After generating the Docker image, the dashboard can be deployed in a Kubernetes cluster through a YAML Ain't Markup Language (YAML) file that will expose the web server outside the cluster, at the Internet Protocol (IP) address of the machine hosting the server in port 31000. The code elements created for this component are available at the repository `https://github.com/Jodao/tma-dashboard`.

As mentioned before in Section 2.4.8, to create the dashboard's interface React and Semantic UI frameworks were used. Although Semantic UI is a Cascading Style Sheet (CSS) framework used for styling, it has a specific library (https://react.semantic-ui.com/) for integrating with React. That library provides React components that can be customized through defined parameters so that a component can be rendered in different ways. Thus, those components, along with others built on my own, were used to assemble the interface. Also, Semantic UI allows theme customization which was used to get components rendered in the same way across the interface.

When a client receives the SPA, it always displays the homepage. To navigate to other sections within the SPA, there is a fixed top bar navigation presented by Fig. 5.6. When it is clicked a new page is rendered according to the icon clicked. Using *react-router-dom* library, this navigation bar and other buttons present in the pages allow the navigation between the following pages, which provide the following functionalities:

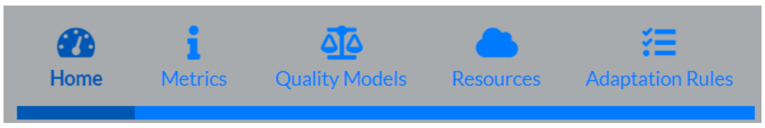


Figure 5.6: Dashboard top navigation bar

- **Homepage**: This is the landing page, presented by Fig. 5.7 of the SPA where configuration files can be imported to add or replace charts up to a maximum of 6. Also, those charts can be removed. The charts on this page will request their data from TMA's API once or continuously (each second) depending on if they are time interval or live plots, respectively;

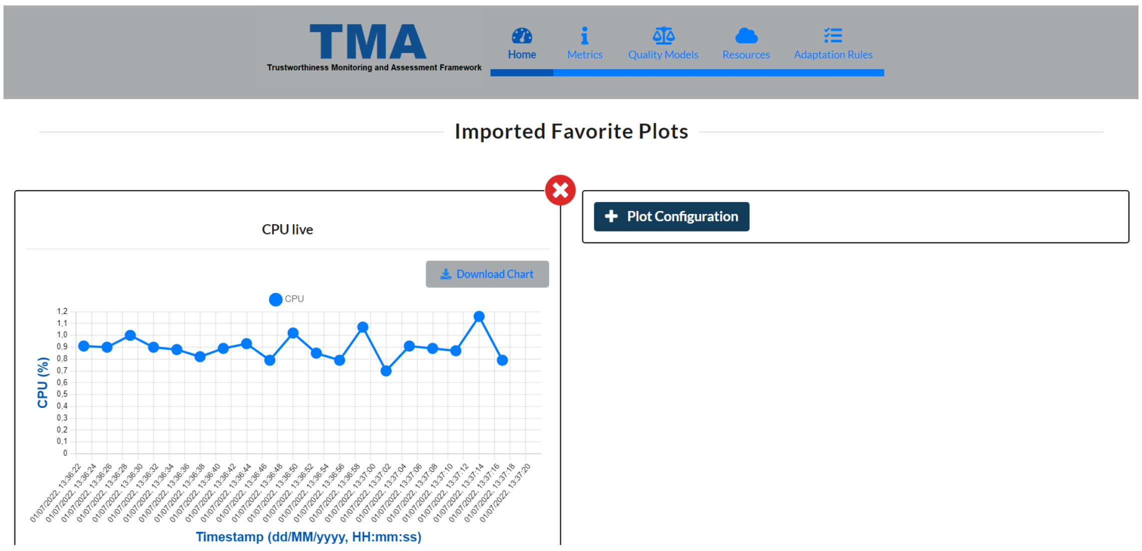


Figure 5.7: Dashboard Homepage

- **Create Rule Page**: On this page, a phased form is presented to create an adaptation rule. Initially, the interface performs a request to the TMA's API to retrieve a list of the resources being monitored and which have actions associated in the database. Then, it presents a DropDown for the user to choose one resource from the list received and to which the adaptation rule is going to be created. Once the choice is confirmed, the interface performs another request to the API to retrieve the weighted metrics tree (configuration profile) associated with the resource.

Then, in the next step of the form, the metrics tree is rendered along with other form fields. These fields request a name for the rule, a DropDown selection of a metric from those on the tree (from which the rule's condition will use the value), a DropDown selection of the condition's operator, and a condition's threshold. Also, there is a preview of the rule condition that gets updated according to inputs. When this step of the form is confirmed, the interface performs another request to the TMA's API to retrieve the actions (and their configurations) associated with the resource chosen initially on the form, and it renders the last phase of the form.

Now, from a DropDown, actions can be selected to incorporate the adaptation plan. Also, to set the execution order of those actions there are buttons to increase/decrease priority. If an action is accidentally added there is a button to remove it from the plan and send it back to the DropDown. If any action requires configurations, they will appear on the interface with the identification of the action they are associated with, and values for them will be required to finish the creation of the rule;

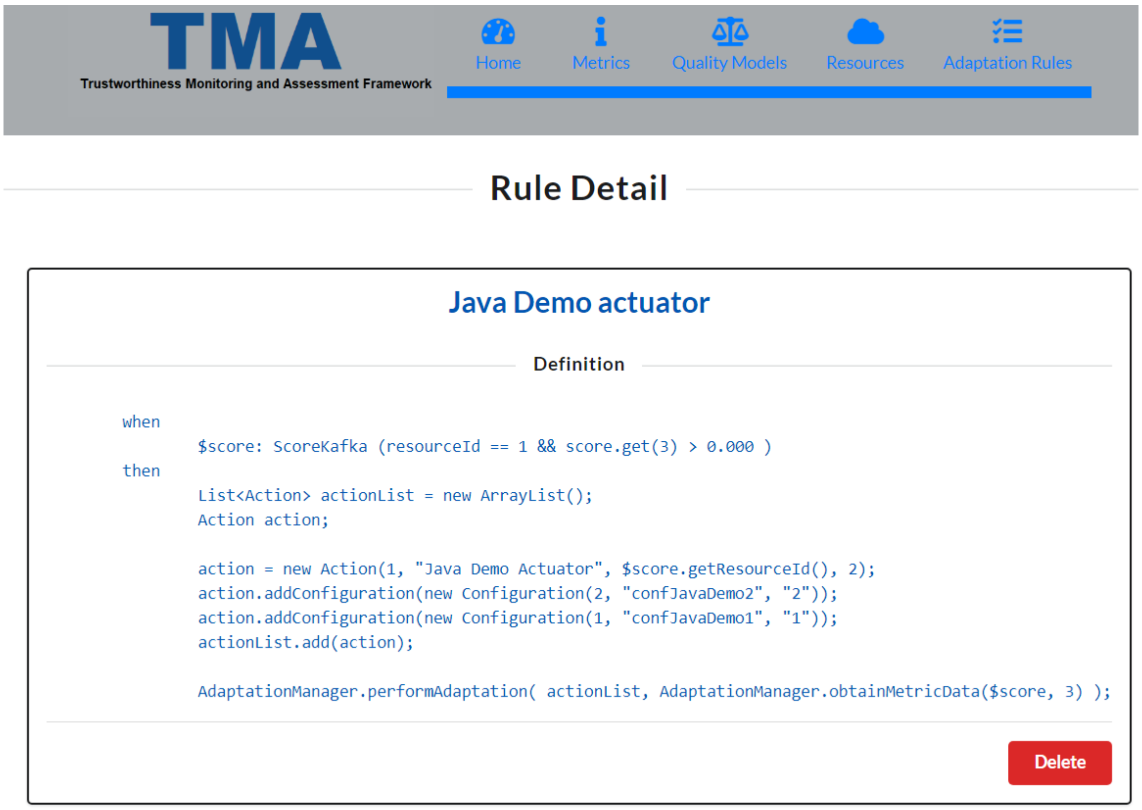
- **List Rules Page**: In this page, initially a request is made to TMA's API to retrieve the list of adaptation rules used by TMA's Planning. Then, that list is rendered as a table along with a text input bar that can be used to filter the results by the rule's name. Here, there is a button that allows navigation to *Create Rule Page* and clicking in a table row navigates to *View Rule Page*;

- **View Rule Page**: In this page a rule's name and code is shown, as depicted by Fig. 5.8, after performing a request for that information to TMA's API. The code consists of the rule's condition and the adaptation plan creation. Also, there is an option to delete the rule;

- **Create Configuration Profile Page**: On this page, it is presented the quality model's (to which the profile is going to be created and associated) id, name, and corresponding metrics tree. Also and alongside, there is a form, shaped like a table, to be filled with the configuration profile's weights to apply on each node of the metrics tree. As weights get filled, the metrics tree gets updated accordingly. When the button to create the configuration profile is pressed, a validation will be performed on the sum of the weights of each parent's child, to verify if it is equal to 1;

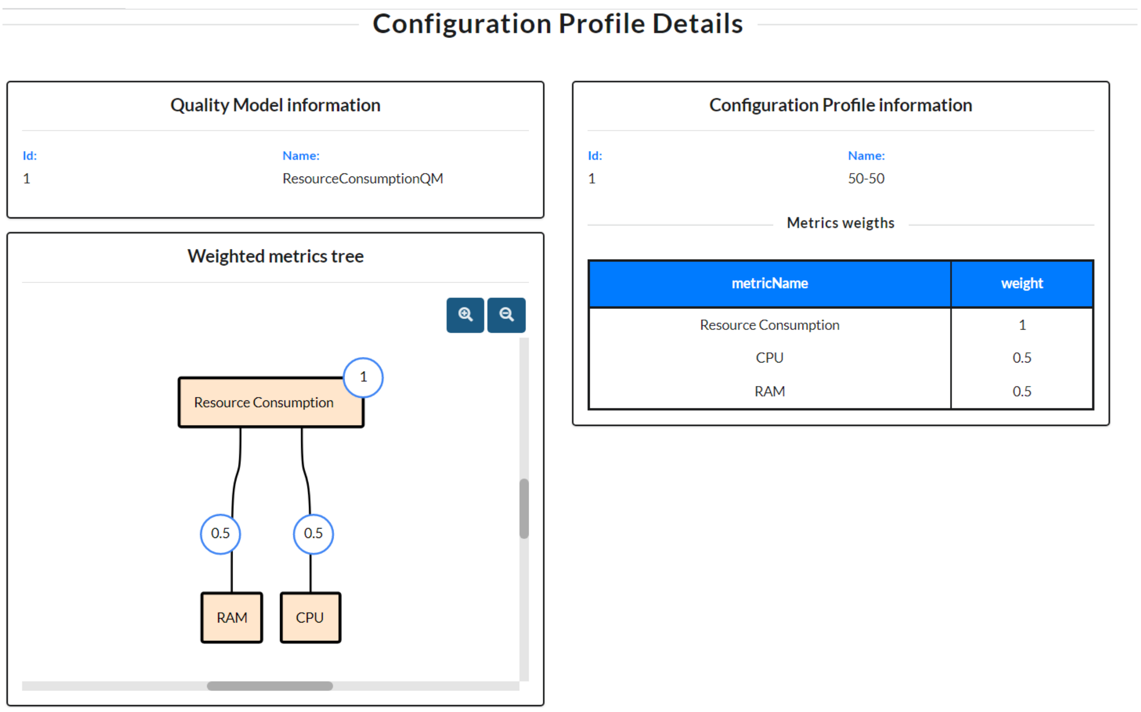
- **View Configuration Profile Page**: This page, shown in Fig. 5.9, presents the id and name of the configuration profile and associated quality model. Also, the profile's weighted metrics tree is shown, and alongside it is presented a table with each tree node's weight;

- **Create Metric Page**: In this page, which is depicted by Fig. 5.10, a form is presented to create a metric. Before rendering the form, requests are sent to TMA's API to retrieve lists of descriptions and metrics that will be necessary to perform associations. There is an option that can be toggled to switch between the form to create a leaf metric and a parent metric. Each



```
when
        $score: ScoreKafka (resourceId == 1 && score.get(3) > 0.000 )
then
        List<Action> actionList = new ArrayList();
        Action action;

        action = new Action(1, "Java Demo Actuator", $score.getResourceId(), 2);
        action.addConfiguration(new Configuration(2, "confJavaDemo2", "2"));
        action.addConfiguration(new Configuration(1, "confJavaDemo1", "1"));
        actionList.add(action);

        AdaptationManager.performAdaptation( actionList, AdaptationManager.obtainMetricData($score, 3) );
```

Figure 5.8: Dashboard's View Rule Page


Configuration Profile Details
Quality Model information
Weighted metrics tree
Resource Consumption
1
0.5
0.5
RAM
CPU
Configuration Profile information
Metrics weigths

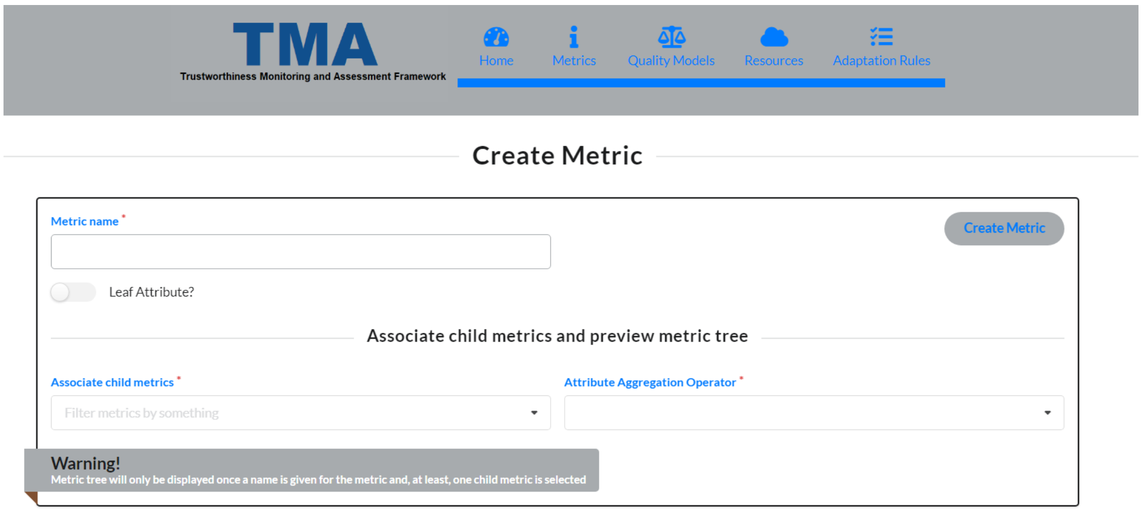
Id:	Name:
1	ResourceConsumptionQM

Id:	Name:
1	50-50

metricName	weight
Resource Consumption	1
CPU	0.5
RAM	0.5

Figure 5.9: Dashboard's View Configuration Profile Page

of the forms is composed of text inputs and DropDown choices. In the leaf metric's form, the fields requested are selections for the description to associate, aggregation operator, normalization method and kind, and the insertion of the number of samples, minimum and maximum thresholds. In the case of creating a parent metric, a metrics tree is generated and updated according to the child metrics chosen. In case the chosen children produce conflicts, the tree is not shown and an informative message appears. For instance, a metric could appear more than once in the tree and that should not be allowed. That would happen if that metric and a parent of it were chosen to be children;


TMA
Trustworthiness Monitoring and Assessment Framework
Home
Metrics
Quality Models
Resources
Adaptation Rules
Create Metric
Metric name *
Create Metric
Leaf Attribute?
Associate child metrics and preview metric tree
Associate child metrics *
Filter metrics by something
Attribute Aggregation Operator *

Warning!
Metric tree will only be displayed once a name is given for the metric and, at least, one child metric is selected

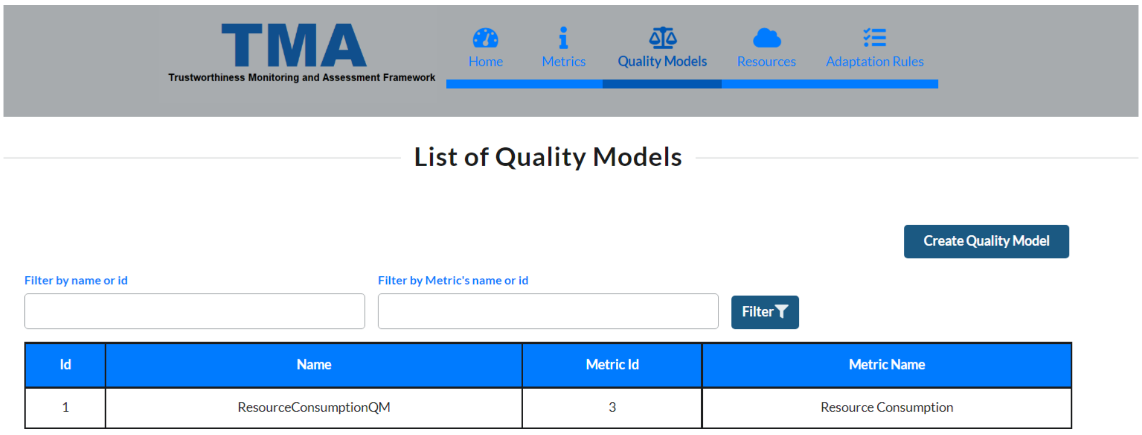
Figure 5.10: Dashboard's Create Metric Page

- **List Metrics Page**: In this page a list of metrics retrieved from TMA's API is rendered as a table along with a text input bar that can be used to filter the results by the metric's id or name. Here there is a button that allows navigation to *Create Metric Page* and clicking in a table row navigates to *View Quality Model Page*;

- **View Metric Page**: This page presents a metric's information depending on if it is a leaf or parent metric. For both, the id, name, and an indication of being or not a leaf metric are shown. But for leaf metrics specific information regarding them (associated description, aggregation operator, number of samples, normalization method and kind, minimum and maximum thresholds) is presented, while for a parent metric its metrics tree is presented;

- **Create Quality Model Page**: On this page, a simple form requesting the name and a metric to associate the quality model is presented. Before rendering the form, a request is made to TMA's API to retrieve the list of metrics that are not associated with a quality model. Also, whenever the metric association is chosen/changed a request is made to TMA's API, to receive the selected metric's tree data to be rendered;

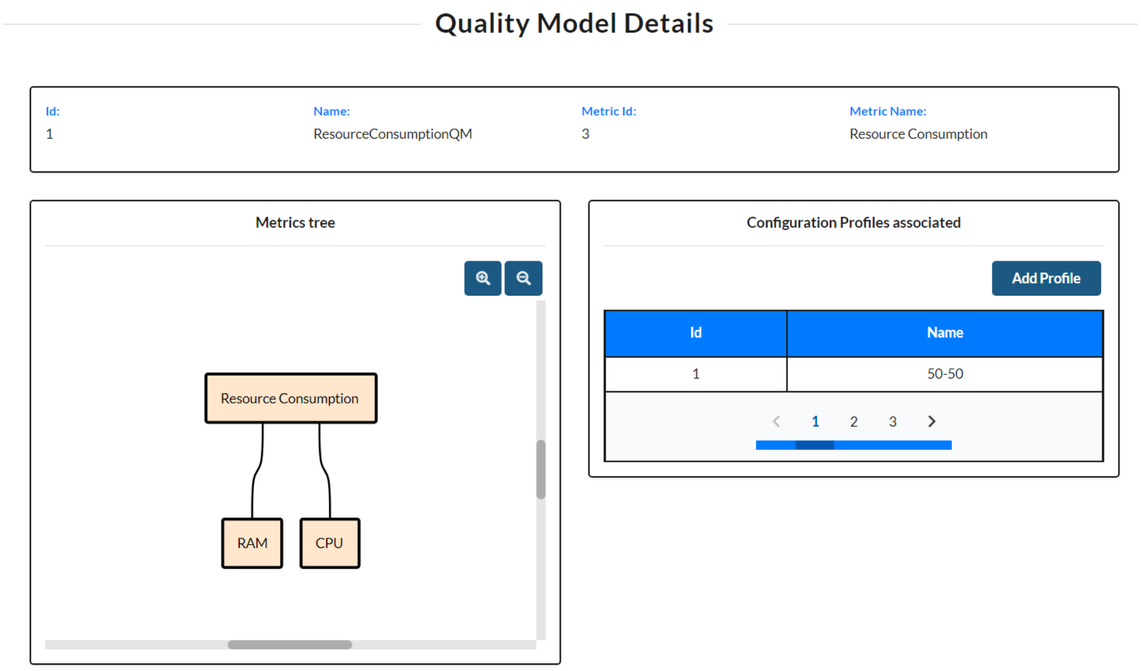
- **List Quality Models Page**: This page, presented by Fig. 5.11, renders a table with the list of quality models retrieved from TMA's API along with two text input bars that can be used to filter the results by metric's id or name and quality model's id or name. Here, there is a button that allows navigation to *Create Quality Model Page* and clicking in a table row navigates to *View Quality Model Page*;



Id	Name	Metric Id	Metric Name
1	ResourceConsumptionQM	3	Resource Consumption

Figure 5.11: Dashboard's List Quality Models Page

- **View Quality Model Page**: This page, shown in Fig. 5.12, presents metadata information regarding the quality model (id and name) and its metrics tree. Also, a list with the configuration profiles associated is presented in the form of a table. From here, there is a button that allows navigation to *Create Configuration Profile Page* and clicking in a table row navigates to *View Configuration Profile Page*;



Id: 1
Name: ResourceConsumptionQM
Metric Id: 3
Metric Name: Resource Consumption

Id	Name
1	50-50

Figure 5.12: Dashboard's List Quality Models Page

- **Plot Resource Metrics Page**: On this page, a phased form is presented to accomplish the goal of generating charts to visualize probed and metric data and occurrences of adaptation plans.

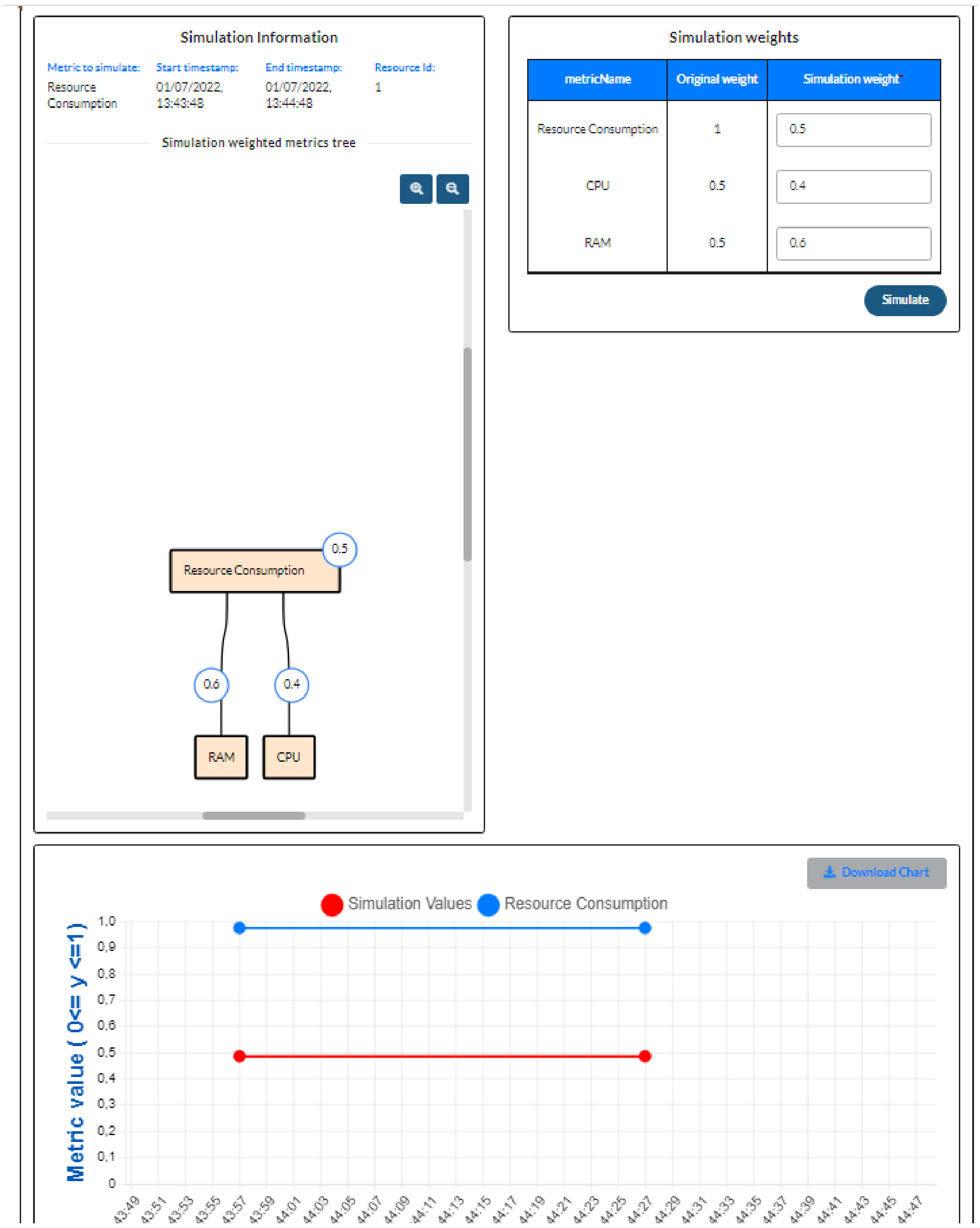
  Initially, the interface performs a request to the TMA's API to retrieve a list of the resources being monitored. Then, it presents a DropDown for the user to choose one resource from the list received and a selection between plotting *raw* or *metric* data. The first is data collected by probes, while the second corresponds to values calculated by TMA's Analyze component. Once the choices are made and confirmed, the interface performs another request to the API to retrieve the weighted metrics tree (configuration profile) associated with the resource.

  Then, in the next step of the form, the metrics tree is rendered along with other form fields. These fields require a selection of a metric from those composing the tree and the indication if a live plot is wanted or not. If not, a timestamp interval must be defined. For the case metric data was chosen at the first phase of the form, additionally, there is an option to add adaptation plans on the chart that can be toggled. As for the raw data option, the metrics allowed to be chosen are the tree's leaves because it is the probed data that will be shown and not the metrics values.

  After confirming the inputs on the second phase of the form, a chart is generated according to the options defined. If the live plot option was chosen then, at each second, a request for the last-minute values will be sent to TMA's API and the chart will get updated. Also, and for the case metric data option was chosen, if there are data values plotted on the chart, a button to navigate to *Simulate Resource Metrics Page* is shown. If there are no data values to be plotted either in the time interval defined or in the live plot, a message to the user is presented. This message alerts that there is no data available, but if the live plot option was chosen and any data arrives in the database, it will be presented on the chart;

- **Simulate Resource Metrics Page**: This page, shown in Fig. 5.13, is only accessible from a plot with metric values in the *Plot Resource Metrics Page* and, here, simulations of metric values can be performed. So, when the user arrives at this page, information regarding the chart he was seeing, which now respects the simulation, is presented, as well as the chart. This information presents the time interval of the simulation, the metric being simulated, and the resource metric data and simulation are associated with. Also, part of the resource's weighted metrics tree, which includes the metric being simulated and its descendants, is presented. There is a table too, that contains the metrics names, original weights, and simulation weights (which initially have values equal to original weights) from that part of the tree.

  The simulation weights are editable, in a form style. After setting the desired weights for the simulation and confirming them, a validation is performed on the parent's child's weights to verify that the sum of the weights is equal to 1. If the weights are valid, then a request is sent to TMA's API to retrieve the simulated metrics values which are added to the chart.



Simulation Information

Metric to simulate:	Start timestamp:	End timestamp:	Resource Id:
Resource Consumption	01/07/2022, 13:43:48	01/07/2022, 13:44:48	1

Simulation weighted metrics tree

Simulation weights

metricName	Original weight	Simulation weight*
Resource Consumption	1	0.5
CPU	0.5	0.4
RAM	0.5	0.6

Figure 5.13: Dashboard's Simulate Resource Metrics Page

In the pages where there are filters, every time they are applied, requests are sent to TMA's API. Also, there is a JavaScript component (**DropDownDataFormat.js** file) that converts data received from the TMA's API into the format acceptable by the DropDown component provided by Semantic UI. The format in which the data is converted depends on the type of entity to be represented. Thus, when a component needs to convert an entity's data to be presented in a DropDown, the corresponding entity's method is invoked from this component.

Whenever there are forms, a component (**ValidInputs.js** file) providing methods to validation of inputs is used. It contains, for instance, validation of strings, DropDown selections, timestamps, and floats. If these inputs are not valid, then a message alerting for their incorrectness is given along with a tip on how to complete them.

To complement these pages, there is a JavaScript component (**ApiModule.js** file) with multiple methods that deal with communications with TMA's API. These component's methods are invoked across the pages by other components to receive and send data to the TMA's API. To ease and deal with the Hypertext Transfer Protocol (HTTP) requests, performed to the TMA's Representational State Transfer (REST) API, the ***axios*** library was used.

Fig. 5.14 presents a sequence diagram that describes the behavior just presented above between a dashboard's page with *ValindInputs* and *ApiModule* when a form is filled and submitted by the user.

To generate charts across the interface a component (**Plot.js** file) was created. This component uses a chart generation library, ***chart.js***, to ease the creation of charts and it adapts the fonts according to the size of the chart. Also, it incorporates a button for exporting the charts as images in PDF files. To generate the PDF files, ***jspdf*** library is used.

Concerning the rendering of metric trees (weighted or not), two components were built (**TreeRender.js** and **TreeLabel.js** files). Along with a diagram creation library, ***reaflow***, *TreeRender* component generates trees where the nodes represent metrics names and the edges may have weights. It also allows to zoom in and zoom out the generated tree and provides means for adding and updating the tree's nodes and edges in forms. *TreeLabel* component is used to override reaflow's default labeling, to allow labeling tree's root node and edges as wanted.

Finally, to optimize the load performance of the SPA, *code splitting* (`https://reactjs.org/docs/code-splitting.html`) was applied to each page. Meaning that each page accessible through navigation will only be loaded when it is needed for the first time.

TMA's API

In this section, a list containing endpoints added to TMA's API and their behavior is presented. This list contains all the endpoints the *Dashboard*, just described in the previous section, uses to provide its functionalities.

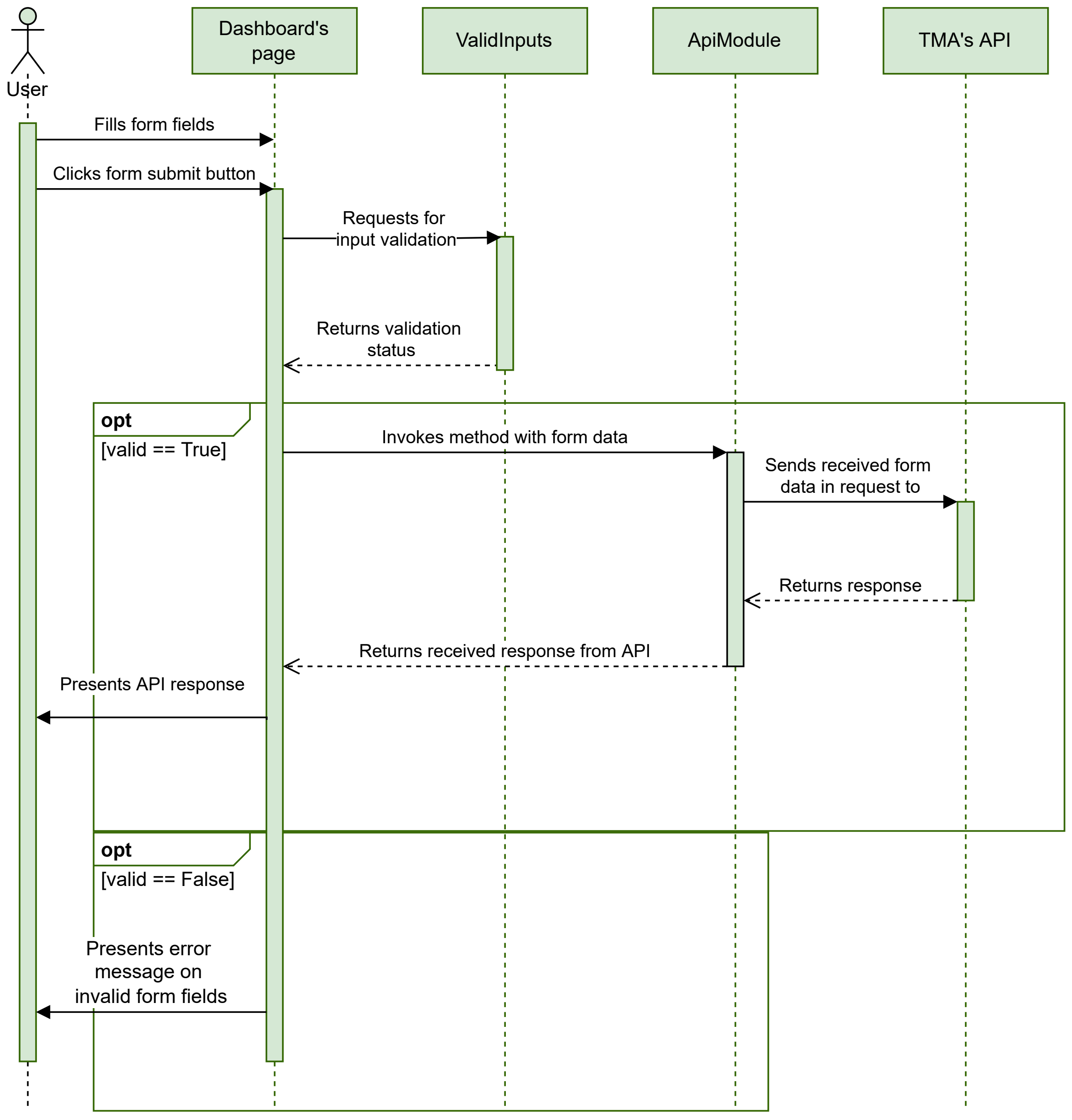

User
Dashboard's page
ValidInputs
ApiModule
TMA's API
Fills form fields
Clicks form submit button
Requests for input validation
Returns validation status
opt
[valid == True]
Invokes method with form data
Sends received form data in request to
Returns response
Returns received response from API
Presents API response
opt
[valid == False]
Presents error message on invalid form fields

Figure 5.14: Sequence diagram for forms submission

To provide its functionalities, TMA's API communicates with TMA's Knowledge database to retrieve data. The added endpoints return the following status codes:

- 200, if the request succeeds, along with a message informing the success or any data necessary to retrieve related to the request;
- 400, if something is wrong with the request such as the parameters received. Also, there are validations performed on some of the endpoints, and, when those validations fail this status code is returned along with a message informing why;
- 500, if there are communication problems internally, either with the database or with TMA's Planning API. When this occurs, a message informing of where the connection failed is returned.

Following the added endpoints are presented and described:

- ***getMetrics***: This is a *GET* endpoint that returns a list of metrics. It can receive **filter** and **createQualityModel** parameters. If a *filter* parameter is received, the metrics returned must match it on their ids or names. If the *createQualityModel* parameter is received, then the metrics to be returned are the ones that are not associated to quality models;
- ***getMetrics/[id]***: This is a *GET* endpoint that returns information regarding a metric. The *[id]* path parameter is the metric's database id. The metric returned can either be a parent metric or a leaf metric. For both, the id and name are retrieved. Concerning a parent metric, additionally, the metrics tree and the children aggregation operator are returned. As for a leaf metric, information about associated raw data and how to process it are returned;
- ***createMetric***: This is a *POST* endpoint that creates a metric. As a payload (illustrated by Fig. 5.15 and 5.16), it receives a metric that contains a name and other pieces of information depending on if the metric is a leaf or a parent. In the case of being a leaf, additionally, it receives an associated description id, a metric aggregation operator, the number of samples, a normalization method, a normalization kind, a minimum threshold, and a maximum threshold. For the case of being a parent, it receives a list of metric ids of its children, and an attribute aggregation operator. In both cases, before the creation, a validation of the uniqueness of the name is made. If this validation fails, a response is returned informing that the name is already in use. Concerning parent metrics, there is also a validation made on the set of child metrics. If there is a metric that is already the parent of the child set received, the response returned informs of that situation, and the parent metric is not created;
- ***getDescriptions***: This is a *GET* endpoint that returns a list of descriptions. It can receive a **filter** parameter, in which case the descriptions returned must match on their ids or names;

```
{
    "metricName": "Leaf",
    "leafAttribute":
    {
        "metricAggregationOperator": 0,
        "numSamples": "10",
        "normalizationMethod": "MIN-MAX",
        "normalizationKind": 1,
        "minimumThreshold": "0",
        "maximumThreshold": "100",
        "description":
        {
            "descriptionId": "1"
        }
    }
}
```

Figure 5.15: Payload of *createMetric* endpoint for the creation of a leaf metric.

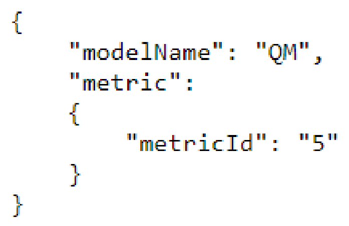
```
{
    "metricName": "Parent",
    "childMetrics":
    [
        {
            "metricId": 4,
        },
        {
            "metricId": 1,
        }
    ],
    "attributeAggregationOperator": 0
}
```

Figure 5.16: Payload of *createMetric* endpoint for the creation of a parent metric.

- ***getQualityModels***: This is a *GET* endpoint that returns a list of quality models. It can receive **qualityModelsFilter** and **metricsFilter** parameters. These parameters are combined to return quality models that either match *qualityModelsFilter* on their ids or names, or have their associated metric's id or name matching *metricsFilter*;

- ***getQualityModels/[id]***: This is a *GET* endpoint that returns information (id, name, associated metrics tree, and a list of the associated configuration profiles) regarding a quality model. The *[id]* path parameter is the quality model's database id;

- ***createQualityModel***: This is *POST* endpoint that creates a quality model. As a payload (illustrated by Fig. 5.17), it receives a quality model with a name and associated metric. Before the creation, a validation of the uniqueness of the name is made. If this validation fails, a response is returned informing that the name is already in use;



```
{
    "modelName": "QM",
    "metric":
    {
        "metricId": "5"
    }
}
```

Figure 5.17: Payload of *createQualityModel* endpoint for the creation of a parent metric.

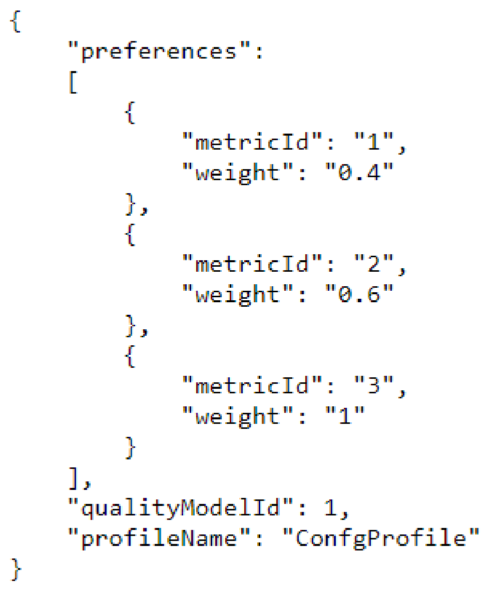
- ***createConfigurationProfile***: This is a *POST* endpoint that creates a configuration profile for a given quality model. As a payload (illustrated by Fig. 5.18), it receives a configuration profile containing a name, the associated quality model id, and a list of metric ids with associated weights. Before the creation, the name for the configuration profile is verified to be unique for the associated quality model. If this validation fails, a response is returned informing that the name is already in use;

- ***getConfigurationProfile/[id]***: This is a *GET* endpoint that returns information (id, name, associated quality model's id, and list of metrics weights) regarding a configuration profile. The *[id]* path parameter is the configuration profile's database id;

- ***getResources***: This is a *GET* endpoint that returns a list of currently monitored resources. It can receive a **createRule** parameter, in which case the resources returned must have actions associated in the database;

- ***getResources/[id]/weightedTree***: This is a *GET* endpoint that returns a resource's associated metrics tree and its weights. The *[id]* path parameter is the resource's database id;

- ***getConfigurationProfile/[id]/listOfMetrics***: This is a *GET* endpoint that returns the list of metrics of a configuration profile. The *[id]* path parameter is a configuration profile's database id;



```
{
    "preferences":
    [
        {
            "metricId": "1",
            "weight": "0.4"
        },
        {
            "metricId": "2",
            "weight": "0.6"
        },
        {
            "metricId": "3",
            "weight": "1"
        }
    ],
    "qualityModelId": 1,
    "profileName": "ConfgProfile"
}
```

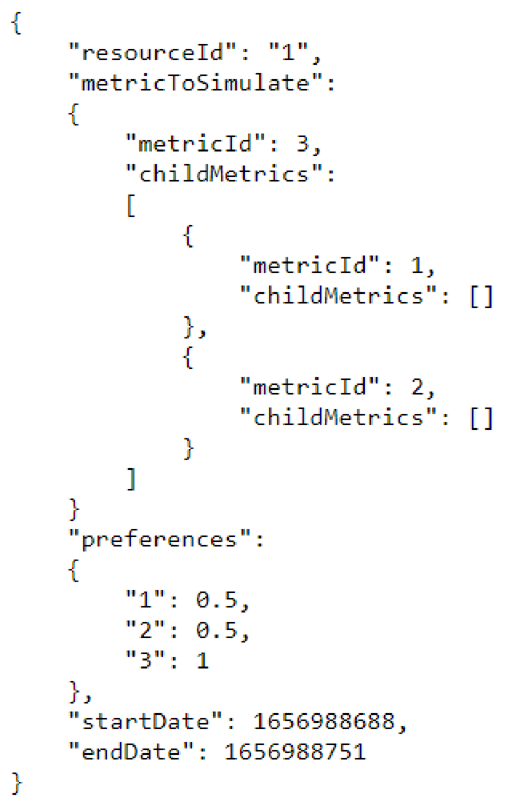
Figure 5.18: Payload of *createConfigurationProfile* endpoint.

- ***getResources/[id]/data***: This is a *GET* endpoint that returns a resource's raw/metric data values along with their timestamp. Optionally, a list of plan ids and the time of occurrence of those plans may be returned. The *[id]* path parameter is the resource's database id. This endpoint receives the parameters **metricId**, **dataType**, **startDate**, **endDate**, **addPlansInfo**. The *metricId* is the id of the metric from which the data will be retrieved. The *dataType* defines if the data values to be returned are raw data collected from probes or metric values calculated by analyze, where the parameter value will be "raw" and "metric" respectively. The parameters *startDate* and *endDate* are epoch values which define the time interval from which the values will be returned. Finally, *addPlansInfo* is a boolean that when set to true will make the endpoint to add a list of plan ids and their timestamps;

- ***simulateData***: This is a *PATCH* endpoint that receives simulation configurations which are applied to calculate and return a list of simulated metric values. To perform the simulation, and as a payload (illustrated by Fig. 5.19), it receives a **resourceId**, a **time interval**, a **metrics tree**, and the list of **simulation weights**. From the *metrics tree* leaves metrics, the *time interval* and the *resourceId*, the raw data from when and from which resource to perform the simulation is identified. Then, applying the received *simulation weights* to the metrics tree nodes, the simulated metric values are returned along with their timestamps;

- ***getPlotsConfigs***: This is a *GET* endpoint that returns a list of plot configurations along with their ids and names;

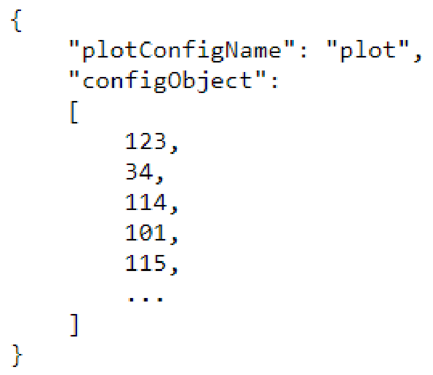
- ***addPlotConfig***: This is a *POST* endpoint that creates and saves a plot configuration in the database. As payload (illustrated by Fig. 5.20), it receives a plot configuration containing a name, and the configuration object of a chart. Before saving the data, a validation is performed on the uniqueness



```
{
    "resourceId": "1",
    "metricToSimulate":
    {
        "metricId": 3,
        "childMetrics":
        [
            {
                "metricId": 1,
                "childMetrics": []
            },
            {
                "metricId": 2,
                "childMetrics": []
            }
        ]
    }
    "preferences":
    {
        "1": 0.5,
        "2": 0.5,
        "3": 1
    },
    "startDate": 1656988688,
    "endDate": 1656988751
}
```

Figure 5.19: Payload of *simulateData* endpoint.

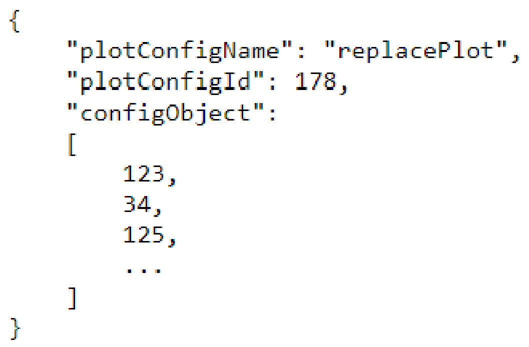
of the name. If this validation fails, a response is returned informing that the name is already in use;



```
{
    "plotConfigName": "plot",
    "configObject":
    [
        123,
        34,
        114,
        101,
        115,
        ...
    ]
}
```

Figure 5.20: Payload of *addPlotConfig* endpoint.

- ***replacePlotConfig***: This is *PUT* endpoint that updates a plot configuration in the database. As payload (illustrated by Fig. 5.21), it receives a plot configuration containing the id of the plot to replace, a name, and a chart's configuration object. The database entity containing the id received gets its information updated. Before replacing a plot's data, a validation is performed on the uniqueness of the received name. If this validation fails, a response is returned informing that the name is already in use;



```
{
    "plotConfigName": "replacePlot",
    "plotConfigId": 178,
    "configObject":
    [
        123,
        34,
        125,
        ...
    ]
}
```

Figure 5.21: Payload of *replacePlotConfig* endpoint.

- ***deletePlotConfig/[id]***: This is a *DELETE* endpoint that deletes a plot configuration. The *[id]* path parameter is the plot configuration's database id to be removed;
- ***getActions***: This is a *GET* endpoint that returns a list of actions and their configurations. It can receive a **resourceId** parameter to retrieve the actions associated to a resource;
- ***"getRules"***: This is a *GET* endpoint that acts as an intermediate. It redirects the request to TMA's Planning API to retrieve the list of adaptation rules. Then, when the response is received, it is returned to the original request. If communication with TMA's Planning fails, a 500 status code is returned informing of that communication problem;
- ***"getRules/[ruleName]"***: This is a *GET* endpoint that acts as an intermediate. It redirects the request to TMA's Planning API to retrieve the code of an adaptation rule. Then, when the response is received, it is returned to the original request. If communication with TMA's Planning fails, a 500 status code is returned informing of that communication problem;
- ***"addRule"***: This is a *POST* endpoint that acts as an intermediate. It redirects the request to TMA's Planning API to create an adaptation rule. Then, when the response is received, it is returned to the original request. If communication with TMA's Planning fails, a 500 status code is returned informing of that communication problem;
- ***"removeRule/[ruleName]"***: This is a *DELETE* endpoint that acts as an intermediate. It redirects the request to TMA's Planning API to remove an adaptation rule. Then, when the response is received, it is returned to the original request. If communication with TMA's Planning fails, a 500 status code is returned informing of that communication problem;

TMA's Analyze

TMA's Analyze is a Java application responsible for applying configuration profiles of quality models on data collected from resources by probes. To recall, quality models represent trees of metrics while configuration profiles the weights to apply on each tree's node. However, before the development of the dashboard, this process was hard-coded, meaning that the management of quality models and their configuration profiles could not be performed without modifying the program.

Therefore, to automate the process of adding/removing resources to monitor, as well as the management of their quality models and configuration profiles, TMA's Analyze implementation was changed.

Now, Analyze processes the raw data collected by probes and calculates the metrics values every 30 seconds. This idle time can be changed if needed. The process to calculate metrics values starts with requests to the database to retrieve the resources being monitored and their quality models and configuration profiles. This is the main change when compared to the previous version: the reading from quality models and their profiles from the database instead of having them hard coded.

When the models have been loaded, Analyze applies to each resource its quality model considering the last 30-second data collected by probes. To calculate the quality model values, the quality model tree is traversed until leaf nodes are reached. For that, the process of going down the tree starts by visiting the root node's children, then the children of its children, and so on. When a leaf is reached, the aggregation and normalization methods and operators, defined in the creation of leaf metrics, are applied to the probed data, and the value obtained is associated with the leaf metric. As these leaves get their values calculated, their parents get their metric values calculated by applying the configuration profile's defined weights. This process is applied until the root node metric is reached and its value also calculated. Finally, after these metric values have been calculated, the data is persisted in the database and the process is restarted for the next resource.

The methods for aggregating and processing data are extensions of *Java abstract classes*. The goal of this approach is to ease the creation and integration of new aggregation and normalization methods and operators.

TMA's Planning

One of the goals of the Dashboard was to allow the management of adaptation rules. Considering that, a few changes had to be made on TMA's Planning which is implemented in Java.

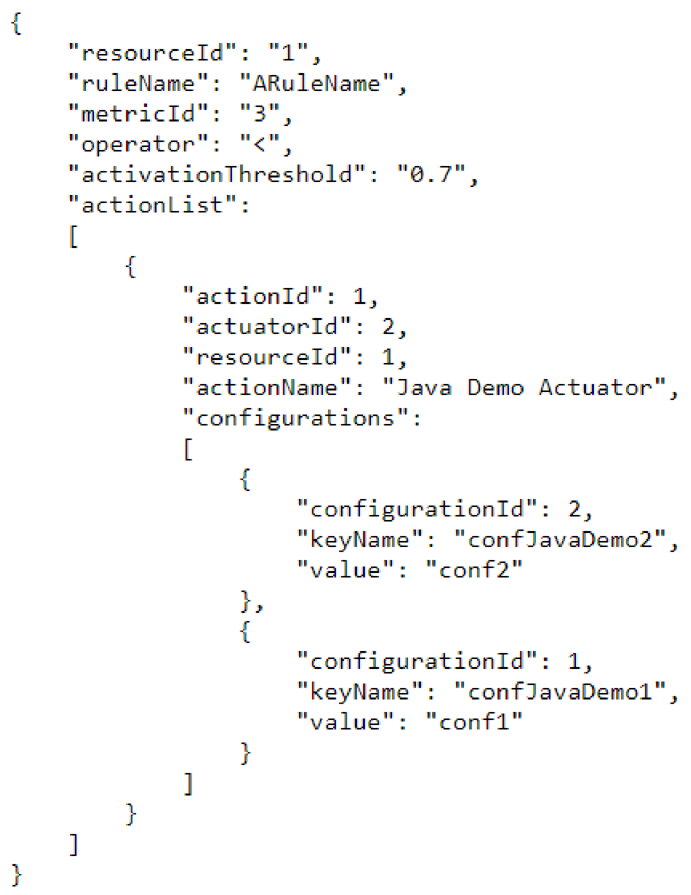
To begin with, the adaptation rules were being read from a file, and, as everything from TMA's operational data is saved in the database, the first thing done was to save and read adaptation rules from a BLOB in the database. Also, TMA's components are meant to run in containers, and, if so, when Planning's container would die, the rules file would be lost. Thus, saving the rules in the database is safer to not lose any configurations.

Considering the goal of managing adaptation rules, a REST API was added to this component using ***Spring Boot***. The existing endpoints and their functions are:

- ***getRules***: This is a *GET* endpoint that returns a list of the adaptation rules names from the database BLOB. It can receive a **filter** parameter, in which case the adaptation rules returned must match their names;

- ***getRules/[ruleName]***: This is a *GET* endpoint that returns the information (name, code's condition and adaptation plan creation) of an adaptation rule from the database BLOB. The *[ruleName]* path parameter is the adaptation rule's name;

- ***addRule***: This is a *POST* endpoint that adds an adaptation rule to the database BLOB. As payload (illustrated by Fig. 5.22), it receives a rule containing a name for the rule, a list of actions and its configurations data for the adaptation plan, and a metricId, resourceId, operator and threshold for defining the rule's condition. Before adding the rule, a validation is performed on the uniqueness of the name. If this validation fails, a response is returned informing that the name is already in use;



```
{
    "resourceId": "1",
    "ruleName": "ARuleName",
    "metricId": "3",
    "operator": "<",
    "activationThreshold": "0.7",
    "actionList":
    [
        {
            "actionId": 1,
            "actuatorId": 2,
            "resourceId": 1,
            "actionName": "Java Demo Actuator",
            "configurations":
            [
                {
                    "configurationId": 2,
                    "keyName": "confJavaDemo2",
                    "value": "conf2"
                },
                {
                    "configurationId": 1,
                    "keyName": "confJavaDemo1",
                    "value": "conf1"
                }
            ]
        }
    ]
}
```

Figure 5.22: Payload of *addRule* endpoint.

- ***removeRule/[ruleName]***: This is a *DELETE* endpoint that removes an adaptation rule from the database BLOB. The *[ruleName]* path parameter is the name of the adaptation rule to delete.

Also, Planning can be seen to be composed of two components: the ***logic*** component and the ***API*** component. Whenever the *API* receives a request that gets completed and updates the adaptation rules database BLOB, the *logic* component is notified that there are updates in the rules.

The *logic* component applies rules to calculated metrics values within a period and, at each period's time, it verifies if it has received a notification. If it has, it reads the database BLOB, updates the adaptation rules set and removes the notification. To produce these notifications a *Java Atomic Boolean* variable is used. This is a variable held by the *logic* component that is passed on the initialization of the *API* component. The use of an Atomic Boolean is to synchronize the access to the variable's value between both components.

Chapter 6

Validation

The goal of this dissertation is to develop a dashboard for Trustworthiness Monitoring & Assessment Framework (TMA) that will aid the management of systems and support the decision process. For that, functional and non-functional requirements were elicited. Thus, it becomes necessary to confirm that what has been implemented complies with those requirements.

For validating what has been developed, validation was performed on implemented functional requirements as well as on the non-functional defined in Section 4.2.

In this chapter, the validation plans and their results will be presented in the following sections. These sections will address the functional requirements and then, the non-functional (Usability, Performance, and Maintainability). Before presenting these validation plans and their results, first, it will be described how validation of TMA was performed.

6.1 TMA validation

In the context of a project, **TalkConnect**, TMA was used to provide self-adaptation to a telecommunications system, specifically the ability to add and remove resources according to demand.

It was decided that a resource consumption model, from which the final version is presented by Fig. 6.1, was going to be used based on Central Processing Unit (CPU), Random Access Memory (RAM) and the number of containers of the critical telecommunication system service. Thus, as the system services were deployed in Docker containers, a probe capable of gathering CPU and RAM usage, and the number of containers from the critical telecommunication service was developed. The code of this probe can be consulted at the repository `https://github.com/nmsa/tma-framework-m/tree/master/development/probes/probe-docker-CPU_MEM_usage`.

At that time, it was needed an actuator to allow adaptations to the system's service. Due to errors found, initially, maintenance had to be made on the actuator's

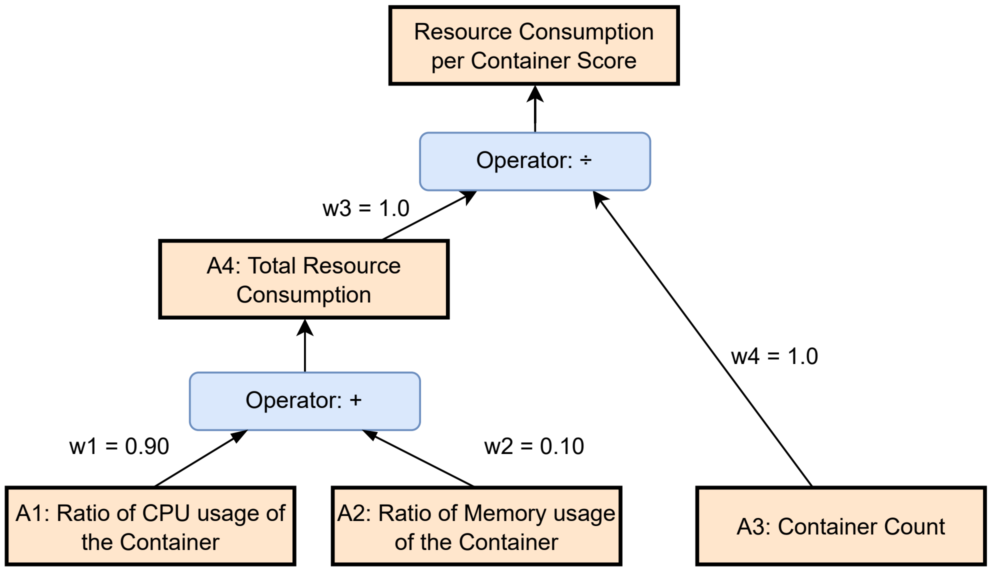

Resource Consumption per Container Score
Operator: ÷
w3 = 1.0
A4: Total Resource Consumption
w4 = 1.0
Operator: +
w1 = 0.90
w2 = 0.10
A1: Ratio of CPU usage of the Container
A2: Ratio of Memory usage of the Container
A3: Container Count

Figure 6.1: TalkConnect project's final quality model.

base code present at the repository `https://github.com/nmsa/tma-framework-e/tree/master/development/libraries/python-actuator-base`. The project partner was responsible for developing the actuator, to which there is no access, based on examples present in a TMA's repository (`https://github.com/nmsa/tma-framework-e/tree/master/development/actuators`).

After the development and integration of the probe and the actuator, some tests were made. First, it was validated that TMA was able to add and remove instances of the telecommunication's system service. Then, load tests were performed to understand the thresholds at which adaptations should be triggered to handle the demand. Finally, and after those adjustments that are presented by the weights in Fig. 6.1, the load tests were repeated and **it was validated that TMA was allowing self-adaptation** on the telecommunication system to meet demand. This means that when the system's service was scaled, the throughput increased.

Thus, it was validated that TMA indeed enables self-adaptation on systems, as long as probes and actuators are created to perform the desired monitoring and adaptations.

6.2 Functional requirements validation

Firstly, this section will present and explain the plan defined to validate the implementation of the functional requirements presented in Section 4.1. Then, the results obtained are presented and analyzed.

6.2.1 Validation plan

To validate the implemented functional requirements, a black box testing approach was used, more precisely equivalence class partitioning and boundary value analysis. Black box is a technique of testing without having any knowledge of the internal working of the application, only examining the fundamental aspects of the system [101]. Equivalence class partitioning splits the range of the input values into equivalent partitions that represent the spectrum of acceptable values, while boundary value analysis tests the boundaries of those partitions [102]. Black box techniques were used because the main product is a user interface and thus, the focus is on testing functionalities.

Since the product developed is a web page, specifically a dashboard, the format of the test cases is somehow different from the ones usually seen when, for example, the outcome of a code function is being validated. Thus, in this specific case, the elaboration of test cases was not straight forward as writing input values because the tests involve actions in the interface.

For defining how to validate what was implemented, the following **test coverage** was applied:

*-All use cases prioritized as **Must** are tested, except for the ones related to the management of adaptation rules. One test case is created for all different types of input classes perceived from the user's perspective. If applicable, boundary test cases will be applied considering the context. Also, where it is applicable, as individual inputs may not be independent of each other, as they are processed together such as in forms, all different combinations of inputs are tested.*

In Table 6.1 the test cases elaborated are presented. For each test case its id, associated use cases ids, input, and expected outcome are shown. One important note for the comprehension of the test cases is that the number of the test case identifies a functionality being tested. For instance, "1.2" and "1.3" represent different functionalities to test. The first number of the test case id represents the category of the test, where the numbers used follow the ones from the Appendix A:

- **1 - Metrics**;
- **2 - Quality Models**;
- **4 - Resources**;
- **11 - Dashboard**.

As for the representation of different input classes and their test cases within a functionality being tested, the test case id should start with the same preceding number. For example "1.2.x.y", where "1.2" represents a functionality being tested in the Metrics category. The ".x" part represents different input classes for that functionality, while the ".y" are test cases for that same input class (defined by ".x"). For example, the test case ids "1.2.1.1" and "1.2.1.2" represent the testing for a given functionality in the Metrics category (initial "1.2" part), where the initial "1.2.1" part represents an input class for that functionality being tested, and "1.2.1.1" and "1.2.1.2" test cases for that input class.

Table 6.1: Functional test cases

Use Case Id	Test Case Id	Input	Expected Outcome
1.1	1.1.1.1	Select a metric that is not a leaf attribute and has a tree height of 1.	Metric's name, id, attribute aggregation operator and indication that the metric is not a leaf attribute are presented, as well as the respective metric tree structure.
1.1	1.1.1.2	Select a metric that is not a leaf attribute and has a tree height of 2.	Metric's name, id, attribute aggregation operator and indication that the metric is not a leaf attribute are presented, as well as the respective metric tree structure.
1.1	1.1.1.3	Select a metric that is not a leaf attribute and has a tree height of 3.	Metric's name, id, attribute aggregation operator and indication that the metric is not a leaf attribute are presented, as well as the respective metric tree structure.
1.1	1.1.2	Select a metric that is a leaf attribute	Metric's name, id, aggregation operator, description associated, number of samples, normalization method, normalization kind, minimum threshold, maximum threshold and an indication that the metric is a leaf attribute are presented.
1.2, 1.7	1.2.1.1	Create a parent metric without filling any field.	Error messages alerting for filling the mandatory fields in which form.
1.2, 1.3, 1.6	1.2.1.2	Create a leaf attribute metric without filling any field.	Error messages alerting for filling the mandatory fields in which form.
1.2, 1.7	1.2.2.1	Create a parent metric without one field, repeating it for each field.	Error message alerting for filling the field that is not filled, and in which form.
1.2, 1.3, 1.6	1.2.2.2	Create a leaf attribute metric without one field, repeating it for each field.	Error message alerting for filling the field that is not filled, and in which form.
1.2, 1.7	1.2.3.1	Create a parent metric after incorrectly filling one field, repeating it for each field.	Error message on the field that is incorrectly filled, presenting information on how to fill it.
1.2, 1.3, 1.6	1.2.3.2	Create a leaf attribute metric after incorrectly filling one field, repeating it for each field.	Error message on the field that is incorrectly filled, presenting information on how to fill it.
1.2	1.2.4.1	Create a parent metric with a name that is already in use by another metric.	A message alerting for the unavailability of the name is presented.
1.2, 1.3	1.2.4.2	Create a leaf attribute metric with a name that is already in use by another metric.	A message alerting for the unavailability of the name is presented.
1.2	1.2.5.1	Create a parent metric filling all the required fields correctly, with 1 child metric	A success message is presented and metric information is correctly inserted in TMA's database.
1.2	1.2.5.2	Create a parent metric filling all the required fields correctly, with 2 child metrics	A success message is presented and metric information is correctly inserted in TMA's database.
1.2	1.2.5.3	Create a parent metric filling all the required fields correctly, with 3 child metrics	A success message is presented and metric information is correctly inserted in TMA's database.
1.2, 1.3	1.2.6.1	Create a leaf attribute metric filling all the required fields correctly.	A success message is presented and metric information is correctly inserted in TMA's database.
1.7, 1.8	1.3.1.1	When creating a parent metric, add and remove child metrics, where the childs have a tree height of 0.	The tree representation is coherent with the selected childs.
1.7, 1.8	1.3.1.2	When creating a parent metric, add and remove child metrics, where the childs have a tree height of 1.	The tree representation is coherent with the selected childs.
1.7, 1.8	1.3.1.3	When creating a parent metric, add and remove child metrics, where the childs have a tree height of 2.	The tree representation is coherent with the selected childs.
1.8, 2.1	2.1.1.1	Select a quality model where its root node has a tree height of 0.	Quality Model's id, name, associated metric, metrics tree structure and list of associated configuration profiles are presented.
1.8, 2.1	2.1.1.2	Select a quality model where its root node has a tree height of 1.	Quality Model's id, name, associated metric, metrics tree structure and list of associated configuration profiles are presented.

Use Case Id	Test Case Id	Input	Expected Outcome
1.8, 2.1	2.1.1.3	Select a quality model where its root node has a tree height of 2.	Quality Model's id, name, associated metric, metrics tree structure and list of associated configuration profiles are presented.
1.8, 2.5	2.2.1.1	Select a configuration profile from a quality model that has a root node with a tree height of 0.	The corresponding configuration profile weights are correctly presented on the quality model's metrics tree.
1.8, 2.5	2.2.1.2	Select a configuration profile from a quality model that has a root node with a tree height of 1.	The corresponding configuration profile weights are correctly presented on the quality model's metrics tree.
1.8, 2.5	2.2.1.3	Select a configuration profile from a quality model that has a root node with a tree height of 2.	The corresponding configuration profile weights are correctly presented on the quality model's metrics tree.
2.6	2.3.1.1	Go to the page for creating a configuration profile for a quality model that has a root node with a tree height of 0.	A form requesting the name and the weights for the quality model's metrics tree nodes is presented.
2.6	2.3.1.2	Go to the page for creating a configuration profile for a quality model that has a root node with a tree height of 1.	A form requesting the name and the weights for the quality model's metrics tree nodes is presented.
2.6	2.3.1.3	Go to the page for creating a configuration profile for a quality model that has a root node with a tree height of 2.	A form requesting the name and the weights for the quality model's metrics tree nodes is presented.
2.6	2.4.1.1	Create a configuration profile without filling any field, for a quality model that has a root node with a tree height of 0.	Error messages alerting for filling the mandatory fields in which form.
2.6	2.4.1.2	Create a configuration profile without filling any field, for a quality model that has a root node with a tree height of 1.	Error messages alerting for filling the mandatory fields in which form.
2.6	2.4.1.3	Create a configuration profile without filling any field, for a quality model that has a root node with a tree height of 2.	Error messages alerting for filling the mandatory fields in which form.
2.6	2.4.2.1	For a quality model that has a root node with a tree height of 0, create a configuration profile without filling one field, repeating it for each field.	Error message alerting for filling the field that is not filled, and in which form.
2.6	2.4.2.2	For a quality model that has a root node with a tree height of 1, create a configuration profile without filling one field, repeating it for each field.	Error message alerting for filling the field that is not filled, and in which form.
2.6	2.4.2.3	For a quality model that has a root node with a tree height of 2, create a configuration profile without filling one field, repeating it for each field.	Error message alerting for filling the field that is not filled, and in which form.
2.6	2.4.3.1	For a quality model that has a root node with a tree height of 0, create a configuration profile after incorrectly filling one field, repeating it for each field.	Error message on the field that is incorrectly filled, presenting information on how to fill it.
2.6	2.4.3.2	For a quality model that has a root node with a tree height of 1, create a configuration profile after incorrectly filling one field, repeating it for each field.	Error message on the field that is incorrectly filled, presenting information on how to fill it.
2.6	2.4.3.3	For a quality model that has a root node with a tree height of 2, create a configuration profile after incorrectly filling one field, repeating it for each field.	Error message on the field that is incorrectly filled, presenting information on how to fill it.
2.6	2.4.4.1	For a quality model that has a root node with a tree height of 1, create a configuration profile where a tree level has the sum of its sibling metrics different than 1, repeating it for the other levels.	A message alerting that a metric's child weights is not equal to 1 is presented
2.6	2.4.4.2	For a quality model that has a root node with a tree height of 2, create a configuration profile where a tree level has the sum of its sibling metrics different than 1, repeating it for the other levels.	A message alerting that the sum of sibling metrics weights is not equal to 1 is presented
2.6	2.4.5.1	Create a configuration profile with a name that is already in use by another configuration profile for the same quality model.	A message alerting for the unavailability of the name is presented.
2.6	2.4.5.2	Create a configuration profile with a name that is already in use by another configuration profile but for another quality model.	A success message is presented and configuration profile information is correctly inserted in TMA's database.
2.6	2.4.6.1	For a quality model that has a root node with a tree height of 0, create a configuration profile filling all the required fields correctly	A success message is presented and configuration profile information is correctly inserted in TMA's database

Use Case Id	Test Case Id	Input	Expected Outcome
2.6	2.4.6.2	For a quality model that has a root node with a tree height of 1, create a configuration profile filling all the required fields correctly	A success message is presented and configuration profile information is correctly inserted in TMA's database
2.6	2.4.6.3	For a quality model that has a root node with a tree height of 2, create a configuration profile filling all the required fields correctly	A success message is presented and configuration profile information is correctly inserted in TMA's database
2.2, 2.9	2.5.1.1	Create a quality model without filling any field.	Error messages alerting for filling the mandatory fields in which form.
2.2, 2.9	2.5.1.2	Create a quality model without filling one field, repeating it for each field.	Error message alerting for filling the field that is not filled, and in which form.
2.2, 2.9	2.5.1.3	Create a quality model after incorrectly filling one field, repeating it for each field.	Error message on the field that is incorrectly filled, presenting information on how to fill it.
2.2, 2.9	2.5.1.4	Create a quality model with a name that is already in use by another quality model.	A message alerting for the unavailability of the name is presented.
2.2	2.5.2.1	Create a quality model filling all the required fields correctly and associating a metric which has a tree height of 0.	A success message is presented and quality model information is correctly inserted in TMA's database
2.2	2.5.2.2	Create a quality model filling all the required fields correctly and associating a metric which has a tree height of 1.	A success message is presented and quality model information is correctly inserted in TMA's database
2.2	2.5.2.3	Create a quality model filling all the required fields correctly and associating a metric which has a tree height of 2.	A success message is presented and quality model information is correctly inserted in TMA's database
1.8, 2.9	2.6.1.1	While creating a quality model, for the metric association select and remove metrics that have a tree height of 0.	The tree representation is coherent with the selected metric.
1.8, 2.9	2.6.1.2	While creating a quality model, for the metric association select and remove metrics that have a tree height of 1.	The tree representation is coherent with the selected metric.
1.8, 2.9	2.6.1.3	While creating a quality model, for the metric association select and remove metrics that have a tree height of 2.	The tree representation is coherent with the selected metric.
1.8, 4.5	4.1	While setting the chart configuration, select a resource and the metric data type.	Associated metrics tree to the resource is presented and the list of selectable metrics match the ones in the tree.
4.5	4.2.1.1	Generate a metric live plot with values available.	Plot is generated with a metric data label and values available are displayed. The plot presents the last minute values. Also, the plot keeps getting updated according to the values being produced.
4.5	4.2.1.2	Generate a metric live plot with no values available.	Plot is generated with a metric data label but no values are displayed. The plot presents the last minute values. Also, the plot keeps getting updated but still with no values being displayed.
4.5	4.2.2.1	Generate a metric time interval plot with values available.	Plot is generated with a metric data label and values available are displayed. The x axis has its values within the time interval.
4.5	4.2.2.2	Generate a metric time interval plot with no values available.	Plot is generated with a metric data label but no values are displayed. The x axis has its values within the time interval.
4.5, 4.7	4.2.3.1	With values available, generate a live plot setting the option to plot plans alongside metric values.	Plot is generated with metric data and plans labels and values available are displayed. Plans data overlap with metrics data and the plot presents the last minute values. Also, the plot keeps getting updated according to the values being produced.
4.5, 4.7	4.2.3.2	With no values available, generate a live plot setting the option to plot plans alongside metric values.	Plot is generated with metric data and plans labels but no values are displayed. Plot presents the last minute values. Also, the plot keeps getting updated but still with no values being displayed.
4.5, 4.7	4.2.4.1	With values available, generate a time interval plot setting the option to plot plans alongside metric values.	Plot is generated with metric data and plans labels and values available are displayed. Plans data overlap with metrics data and the x axis has its values within the time interval.
4.5, 4.7	4.2.4.2	With no values available, generate a time interval plot setting the option to plot plans alongside metric values.	Plot is generated with metric data and plans labels but no values are displayed. Plans data overlap with metrics data and the x axis has its values within the time interval.

Use Case Id	Test Case Id	Input	Expected Outcome
4.8	4.3.1.1	Export a time interval plot with metrics values.	A pdf file is created with the generated plot.
4.8	4.3.1.2	Export a time interval plot with metrics and plans values.	A pdf file is created with the generated plot.
4.8	4.3.2.1	Export a live plot as soon as the plot is generated.	A pdf file is created with the plot at the moment the export button was pressed.
4.8	4.3.2.2	Export a live plot $\sim$10 seconds after the plot is generated.	A pdf file is created with the plot at the moment the export button was pressed.
4.10	4.4.1	Export a plot configuration from a live plot.	A text file is generated containing the corresponding plot configurations. The configuration "livePlot" is set to true.
4.10	4.4.2	Export a plot configuration from a time interval plot.	A text file is generated containing the corresponding plot configurations. The configuration "livePlot" is set to false, while the "startDate" and "endDate" properties are equal to the time interval defined to generate the plot being exported.
4.6	4.5.1	Go to the simulation page.	Information regarding the simulation is presented: configuration profile weights, resource, time interval and a plot with original metrics values. Also, there is a form for filling the simulation weights.
4.6	4.6.1	On the simulation page, fill the weights in the simulation form.	The simulation configuration profile tree representation gets updated with the weights being introduced.
4.6	4.7.1.1	For a configuration profile of a quality model that has a root node of height 0, perform simulation without filling any field.	Error messages alerting for filling the mandatory fields in which form.
4.6	4.7.1.2	For a configuration profile of a quality model that has a root node of height 1, perform simulation without filling any field.	Error messages alerting for filling the mandatory fields in which form.
4.6	4.7.2.1	For a configuration profile of a quality model that has a root node of height 0, perform simulation without filling one field, repeating it for each field.	Error message alerting for filling the field that is not filled, and in which form.
4.6	4.7.2.2	For a configuration profile of a quality model that has a root node of height 1, perform simulation without filling one field, repeating it for each field.	Error message alerting for filling the field that is not filled, and in which form.
4.6	4.7.3.1	For a configuration profile of a quality model that has a root node of height 0, perform simulation after incorrectly filling one field, repeating it for each field.	Error message on the field that is incorrectly filled, presenting information on how to fill it.
4.6	4.7.3.2	For a configuration profile of a quality model that has a root node of height 1, perform simulation after incorrectly filling one field, repeating it for each field.	Error message on the field that is incorrectly filled, presenting information on how to fill it.
4.6	4.7.4.1	For a configuration profile of a quality model that has a root node of height 1, perform simulation where a tree level has the sum of its sibling metrics different than 1, repeating it for the other levels.	A message alerting that the sum of sibling metrics weights is not equal to 1 is presented
4.6	4.7.4.2	For a configuration profile of a quality model that has a root node of height 2, perform simulation where a tree level has the sum of its sibling metrics different than 1, repeating it for the other levels.	A message alerting that the sum of sibling metrics weights is not equal to 1 is presented
4.6	4.7.5.1	For a configuration profile of a quality model that has a root node of height 0, perform simulation filling all the required fields correctly.	The simulation plot gets updated with the simulation data series alongside the original values.
4.6	4.7.5.2	For a configuration profile of a quality model that has a root node of height 1, perform simulation filling all the required fields correctly	The simulation plot gets updated with the simulation data series alongside the original values.
4.6	4.7.6	After performing a simulation, rechange some of the weights and perform a new simulation	The simulation data series gets updated on the plot according to the new weights.
11.1	11.1	Select option to add chart to the homepage	Form to select the location of the configuration file
11.1	11.2.1.1	Adding a chart to the homepage, insert a name that already exists for another chart configuration.	A message alerting for the unavailability of the name is presented.
11.1	11.2.1.2	Adding a chart to the homepage, don't introduce or incorrectly fill the name for the plot configuration.	Error message alerting for filling the mandatory fields in which form.

Use Case Id	Test Case Id	Input	Expected Outcome
11.1	11.2.2.1	Add a live chart to the homepage.	Everytime the Homepage is loaded, the added chart appears and the chart keeps getting updated each second.
11.1	11.2.2.2	Add a time interval chart to the homepage.	Everytime the Homepage is loaded, the added chart appears.
11.2	11.3.1	Select the option to Replace a chart and Select option to replace a chart and cancel the operation	There is no chart replacement.
11.2	11.3.2.1	Replace a chart configuration, inserting a name that already exists for another chart configuration.	A message alerting for the unavailability of the name is presented.
11.2	11.3.2.2	Replace a chart configuration, giving it the name of the chart being replaced.	A message alerting for the unavailability of the name is presented.
11.2	11.3.2.3	Replace a chart configuration, not inserting the name.	Error message alerting for filling the mandatory fields in which form.
11.2	11.3.3.1	Replace a live chart by time interval chart.	The corresponding chart slot on the homepage renders a new chart that is not getting updated every second.
11.2	11.3.3.2	Replace a time interval chart by a live chart.	The corresponding chart slot on the homepage renders a new chart that gets updated each second.
11.3	11.4.1.1	Delete a time interval chart in the first slot, where there are at least 2 charts on the Homepage.	The chart gets deleted from the Homepage and the charts after it are relocated to the previous slot.
11.3	11.4.1.2	Delete a live chart in the first slot, where there are at least 2 charts on the Homepage.	The chart gets deleted from the Homepage and the charts after it are relocated to the previous slot.
11.3	11.4.2.1	Delete a time interval chart between two other charts.	The chart gets deleted from the Homepage and the charts after it are relocated to the previous slot.
11.3	11.4.2.2	Delete a live chart between two other charts.	The chart gets deleted from the Homepage and the charts after it are relocated to the previous slot.
11.3	11.4.3.1	Delete a time interval chart in the last slot, where there are at least 2 charts on the Homepage.	The chart gets deleted from the Homepage and the charts after it are relocated to the previous slot.
11.3	11.4.3.2	Delete a live chart in the last slot where there are at least 2 charts on the Homepage.	The chart gets deleted from the Homepage and the charts after it are relocated to the previous slot.
11.3	11.4.4.1	Delete the only chart of the homepage, where that chart is a time interval one.	No charts remain on the Homepage and there is a button to add a chart.
11.3	11.4.4.2	Delete the only chart of the homepage, where that chart is a live one.	No charts remain on the Homepage and there is a button to add a chart. Also, requests stop being sent to TMA's API.

6.2.2 Results

All the test cases planned in the previous section succeeded as the results from the tests, presented in Table 6.2, show. This means that within what has been tested, the dashboard is functionally valid.

Table 6.2: Functional test cases results

Test Case Id	Pass / Fail
1.1.1.1	Pass
1.1.1.2	Pass
1.1.1.3	Pass
1.1.2	Pass

Test Case Id	Pass / Fail
1.2.1.1	Pass
1.2.1.2	Pass
1.2.2.1	Pass
1.2.2.2	Pass
1.2.3.1	Pass
1.2.3.2	Pass
1.2.4.1	Pass
1.2.4.2	Pass
1.2.5.1	Pass
1.2.5.2	Pass
1.2.5.3	Pass
1.2.6.1	Pass
1.3.1.1	Pass
1.3.1.2	Pass
1.3.1.3	Pass
2.1.1.1	Pass
2.1.1.2	Pass
2.1.1.3	Pass
2.2.1.1	Pass
2.2.1.2	Pass
2.2.1.3	Pass
2.3.1.1	Pass
2.3.1.2	Pass
2.3.1.3	Pass
2.4.1.1	Pass
2.4.1.2	Pass
2.4.1.3	Pass
2.4.2.1	Pass
2.4.2.2	Pass
2.4.2.3	Pass
2.4.3.1	Pass
2.4.3.2	Pass
2.4.3.3	Pass
2.4.4.1	Pass
2.4.4.2	Pass
2.4.5.1	Pass
2.4.5.2	Pass
2.4.6.1	Pass
2.4.6.2	Pass
2.4.6.3	Pass
2.5.1.1	Pass
2.5.1.2	Pass
2.5.1.3	Pass
2.5.1.4	Pass
2.5.2.1	Pass
2.5.2.2	Pass
2.5.2.3	Pass
2.6.1.1	Pass
2.6.1.2	Pass

Test Case Id	Pass / Fail
2.6.1.3	Pass
4.1	Pass
4.2.1.1	Pass
4.2.1.2	Pass
4.2.2.1	Pass
4.2.2.2	Pass
4.2.3.1	Pass
4.2.3.2	Pass
4.2.4.1	Pass
4.2.4.2	Pass
4.3.1.1	Pass
4.3.1.2	Pass
4.3.2.1	Pass
4.3.2.2	Pass
4.4.1	Pass
4.4.2	Pass
4.5.1	Pass
4.6.1	Pass
4.7.1.1	Pass
4.7.1.2	Pass
4.7.2.1	Pass
4.7.2.2	Pass
4.7.3.1	Pass
4.7.3.2	Pass
4.7.4.1	Pass
4.7.4.2	Pass
4.7.5.1	Pass
4.7.5.2	Pass
4.7.6	Pass
11.1	Pass
11.2.1.1	Pass
11.2.1.2	Pass
11.2.2.1	Pass
11.2.2.2	Pass
11.3.1	Pass
11.3.2.1	Pass
11.3.2.2	Pass
11.3.2.3	Pass
11.3.3.1	Pass
11.3.3.2	Pass
11.4.1.1	Pass
11.4.1.2	Pass
11.4.2.1	Pass
11.4.2.2	Pass
11.4.3.1	Pass
11.4.3.2	Pass
11.4.4.1	Pass
11.4.4.2	Pass

During development, a lot of informal testing was done, based on intuition and internal knowledge of the components' behavior and code. Thereby, some of the test cases described and executed, for example, the forms testing, had already been tested before.

6.3 Usability validation

This section presents and explains, primarily, the validation plan for the usability requirements presented in Section 4.2. Then, the results obtained are presented and analyzed.

6.3.1 Validation plan

Usability requirements were elicited concerning tasks that require some effort and attention from users, such as simulating metrics, plotting metrics, and management of the Homepage plots. The focus of usability on these tasks is due to the importance of those features in the product value. Thus, the goal is to find out if the positioning of page elements within the dashboard should be rethought, redesigned, and reimplemented, or if their current placement is good enough for allowing future users to properly manage systems.

The dashboard is a very restrictive product meant to be used by technical people that have a generalized understanding of TMA. Thereby the tests are applied to a single user profile and they would be conducted with a very small group of people, in this case, work colleagues that have a minimal understanding of TMA and how it works. Therefore, the **user profile** defined for these usability tests is:

- **Technological individual**;
- **Comprehension of TMA's goal**;
- **Knowledge of how TMA works at a high level concept**.

Qualitative and quantitative assessments were made on usability. For that, a set of tasks to perform was created for the usability tests considering the following categories:

1. **Plot metric data (and plan data)**;
2. **Simulate metrics data**;
3. **Management of Homepage plots**.

The defined set of tasks is presented in Table 6.3 where each task has an id associated with one of the categories above, a brief description of what the task is about, notes alerting the moderator for something needed during the task, the minimal execution of steps and the expected max duration. These last two properties were assigned to aid in the quantitative evaluation. The task execution is sequential, meaning that they must be executed one after another. This is mandatory because the defined minimal steps needed to be taken to complete the tasks have the sequential execution of tasks in mind.

Table 6.3: Usability test tasks

Id	**Description**	**Notes**	**Minimal Execution of Steps**	**Expected max duration (seconds)**
1.1	Plot raw data in a defined interval	Data generated previously and interval told by moderator	1. Press “Resources” on navigation bar 2. Select a Resource and data type “Raw Data” option 3. Press "Confirm" 4. Select any metric to visualize data 5. Insert start and end timestamp 6. Press "Confirm"	150
1.2	Plot metric data in a defined interval	Data generated previously and interval told by moderator	1. Press the top bar on the “1st step” of the form 2. Select metric data type 3. Press “Confirm” 4. Select which metric to visualize data 5. Insert start and end timestamp 6. Press “Confirm”	120
1.3	Plot live metric data	No data available	1. Press the top bar on the “2nd step” of the form 2. Enable the option of live plot 3. Press "Confirm"	60
1.4	Plot live metric data along with plans	No data available. When completing this task start probe.	1. Press the top bar on the “2nd step” of the form 2. Enable the option to plot plans 3. Press "Confirm"	30
1.5	Export plot config		1. Press button to export plot config	15
1.6	Download chart		1. Press button to download chart	15

Id	Description	Notes	Minimal Execution of Steps	Expected max duration (seconds)
2.1	Go to simulation page	Previously started probe at task 1.4	1. Press button to simulate metrics	15
2.2	Insert simulation weights and calculate simulation		1. Insert simulation weights 2. Press "Simulate" button	60
3.1	Add a plot to the homepage	File generated on Task 1.5 must be used.	1. Press "Home" on navigation bar 2. Press "+ Plot configuration" button 3. Select exported configuration file 4. Insert nonexistent plot configuration name 5. Press the button to save	60
3.2	Replace previously added plot configuration with the same configuration file but edit it to not add plans in the plot		1. Edit configuration file, changing "addPlansInfo" configuration to false 2. Press "Replace configuration" button 3. Select the edited file 4. Insert new name 5. Press "replace" button	60
3.3	Remove added and then replaced plot config from Homepage		1. Press red "X" button	20

Also, during the tests, in some of the tasks questions would be asked to the users. These questions are present in Table 6.4 with an id, the associated task id, the question itself, and the answer. The goal was to understand if the interface provided sufficient feedback and information for the users to understand what was happening.

Table 6.4: Usability test questions

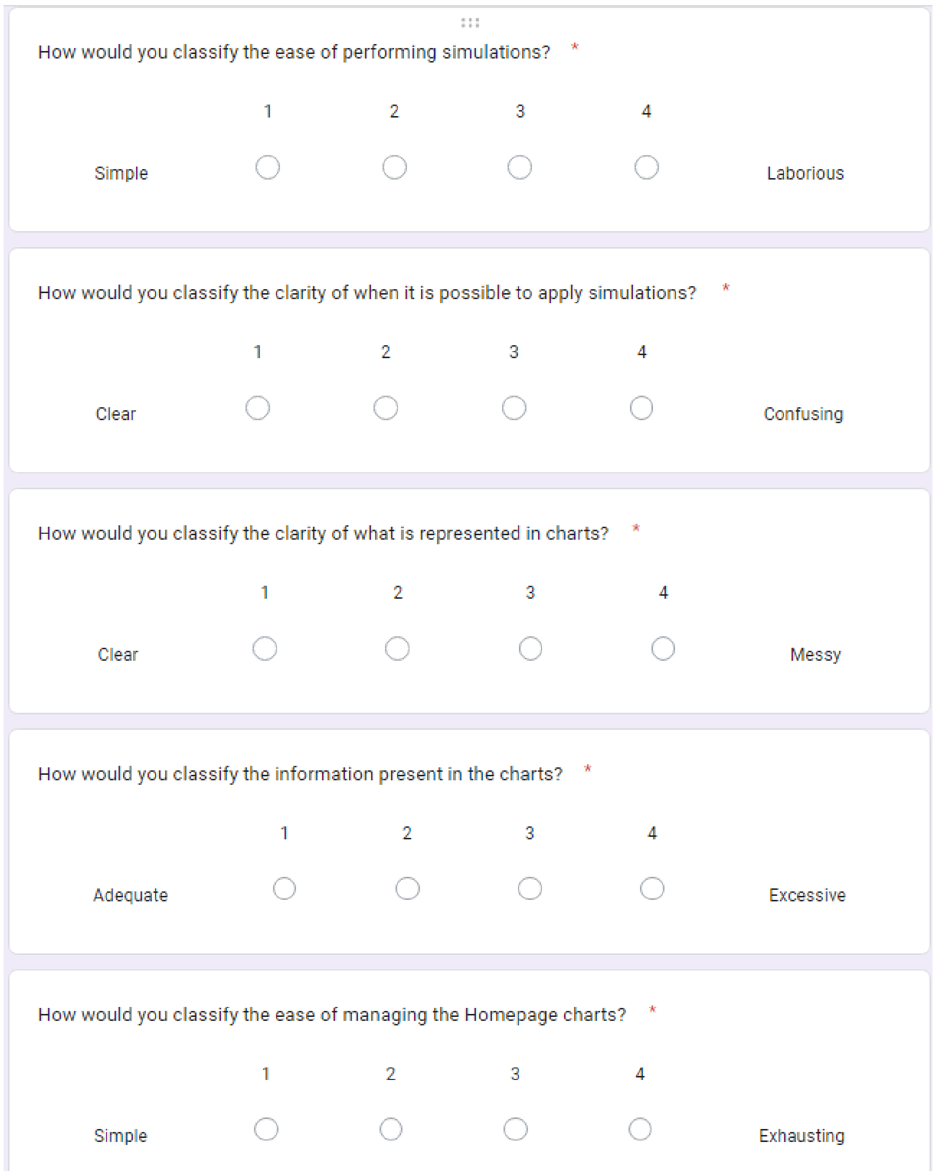
Question Id	Task Id	Question	Answer
1	1.3	Can you tell me what is happening? Why are there no points plotted?	There is no data available. Also, that information is presented on the interface.
2	1.4	Is there any difference to the previous plot?	Yes, there is a label for the plans data series.
3	2.1	Can you tell me when you can apply simulations?	Only when there is data available whether live or in a time interval plot. And applied to metric data only, not raw data.
4	2.1	Can you tell me when and to what the simulation is going to be applied?	Must indicate the simulation parameters that appear on the top of the page.
5	2.2	What happened after you pressed the button to simulate?	The chart got updated with a new series that corresponds to the simulated values using the inserted weights.

Question Id	Task Id	Question	Answer
6	3.2	Can you tell me what constitutes the chart configuration file and the types of data associated to each parameter in that file?	• **resourceId**: number that identifies the resource; • **dataType**: string that identifies if the chart values are raw or metric data; • **metricId**: number that identifies the metric; • **startDate**: timestamp in epoch that defines the start of the time interval from which values are presented in the chart; • **endDate**: timestamp in epoch that defines the end of the time interval from which values are presented in the chart; • **livePlot**: boolean that defines if the chart will keep updating its values; • **addPlansInfo**: boolean that defines if plans series is presented on the chart; • **metricLabel**: string to be presented on the chart's y axis label.

At the end of the test, and offline, users would also complete an anonymous form to provide a qualitative evaluation of the product. The offline and anonymity purposes are to not influence the test users when giving their opinion about the product's functionalities they just tested. The form is composed of 5 semantic differential scale questions, depicted by Fig. 6.2, and 5 Likert scale questions, presented by Fig. 6.3. Likert scales are useful for stating how much people agree or disagree with a particular statement, while semantic differential scales are used for people to decide how much of a trait or quality an item has for them [103]. In both types of questions, an even number of possible answers were set to force their decision to be negative or positive. Thus, there can not be neutral positions.

Considering the usability requirements defined in Section 4.2, for a first interaction with the dashboard, the metrics to validate within these usability tests are:

1. Every user completes 80 % of the planned tasks.
2. There is a maximum of 3 errors in completed tasks;
3. Tasks do not take longer than the expected maximum duration;
4. 75 % of the users can correctly answer question 3;
5. 75 % of the users can correctly answer question 4;
6. 100 % of the users can correctly identify 6 out of the 8 parameters of question 6.
7. An average of 75 % positive answers in offline form's questions is received.



How would you classify the ease of performing simulations? *

	1	2	3	4	
Simple	○	○	○	○	Laborious

How would you classify the clarity of when it is possible to apply simulations? *

	1	2	3	4	
Clear	○	○	○	○	Confusing

How would you classify the clarity of what is represented in charts? *

	1	2	3	4	
Clear	○	○	○	○	Messy

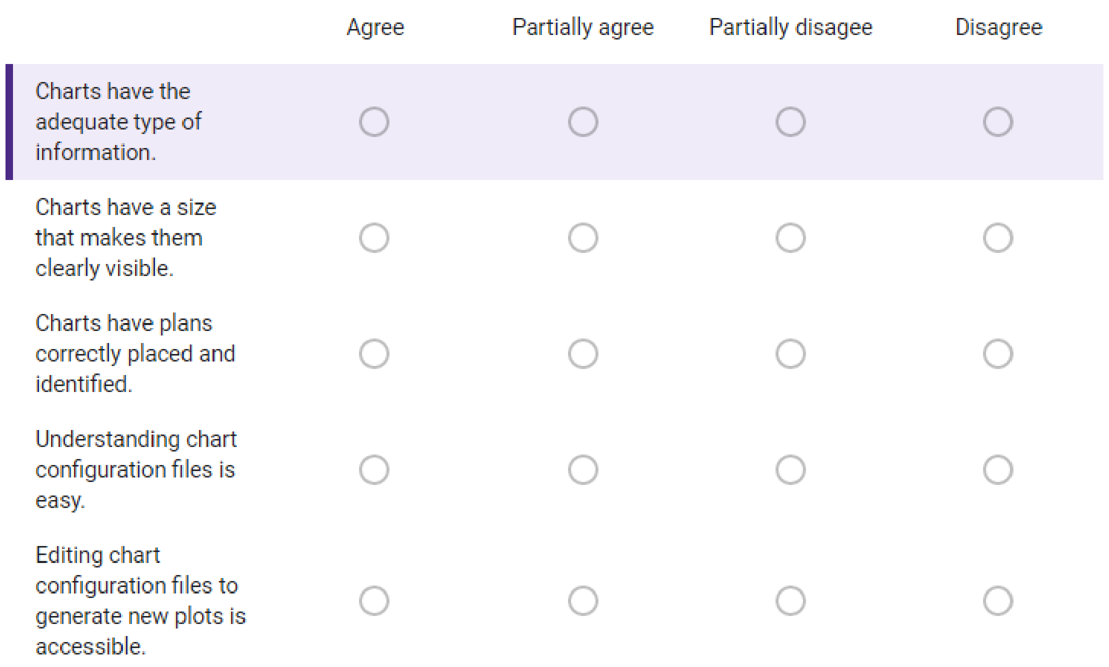
How would you classify the information present in the charts? *

	1	2	3	4	
Adequate	○	○	○	○	Excessive

How would you classify the ease of managing the Homepage charts? *

	1	2	3	4	
Simple	○	○	○	○	Exhausting

Figure 6.2: Usability test form's semantic differential questions.



	Agree	Partially agree	Partially disagee	Disagree
Charts have the adequate type of information.	○	○	○	○
Charts have a size that makes them clearly visible.	○	○	○	○
Charts have plans correctly placed and identified.	○	○	○	○
Understanding chart configuration files is easy.	○	○	○	○
Editing chart configuration files to generate new plots is accessible.	○	○	○	○

Figure 6.3: Usability test form's likert scale questions.

Other tasks and questions are not considered in the metrics above, but they would be analyzed as well. That is done because there may be other problems in the interface that were not the focus of these tests, but could be detected during the tests.

Before executing the tests some conditions must be assured:

- Create a quality model of Resource consumption, based on CPU and RAM, to be considered during and presented before starting the tests;
- Have previously generated data for the quality model;
- Have a probe ready to start collecting data from a system and a demo actuator running;
- Define a dummy adaptation rule to keep triggering the creation of plans to invoke a demo actuator.

The tests would be moderated and conducted remotely with a total of 4 users, recording the sessions for data gathering and analysis purposes.

6.3.2 Results

The usability tests were run with a total of 4 users and the sessions were recorded for posterior analysis and data collection. The results concerning the quantitative evaluation are presented in Table 6.5 and Table 6.6, which respect to tasks and questions respectively.

Table 6.5: Usability tasks results

Task Id	User Id	Duration (s)	Nº Extra steps	Nº Errors	Completed	Notes
1.1	1	70	0	0	yes	
	2	52	0	1	yes	Error inserting the form timestamps, but the product alerted for that.
	3	57	0	0	yes	
	4	110	0	0	yes	Unlike the other users that were relatively fast for a first interaction, this one took a longer time to complete the task. But that was already expected since it was the first interaction with the product. As a consequence he navigated a lot through the interface to understand what to do.
1.2	1	50	0	0	yes	
	2	60	1	0	yes	The user did an extra step because he did not even try to use form steps to go back.
	3	37	1	0	yes	The user did an extra step because he did not even try to use form steps to go back.
	4	59	1	0	yes	The user did an extra step because he did not even try to use form steps to go back.
1.3	1	5	0	0	yes	
	2	12	0	0	yes	The user tried to use the form steps, which worked.
	3	10	1	0	yes	The user did an extra step because he did not even try to use form steps to go back.
	4	14	1	0	yes	The user did an extra step because he did not even try to use form steps to go back.
1.4	1	15	0	0	yes	
	2	5	0	0	yes	
	3	9	1	0	yes	The user did an extra step because he did not even try to use form steps to go back.
	4	12	1	0	yes	The user did an extra step because he did not even try to use form steps to go back.
1.5	1	3	0	0	yes	
	2	3	0	0	yes	
	3	2	0	0	yes	
	4	5	0	0	yes	
1.6	1	2	0	0	yes	
	2	4	0	0	yes	
	3	2	0	0	yes	
	4	5	0	0	yes	
2.1	1	4	0	0	yes	
	2	7	0	0	yes	
	3	2	0	0	yes	
	4	9	0	0	yes	
2.2	1	27	0	0	yes	
	2	12	0	0	yes	
	3	20	0	1	yes	Tried to insert weights for a parent metric's childs that were not equal to 1. The interface alerted for that.
	4	21	0	1	yes	Tried to insert weights for a parent metric's childs that were not equal to 1. The interface alerted for that.
3.1	1	17	0	0	yes	
	2	40	0	0	yes	
	3	27	0	0	yes	
	4	25	0	0	yes	
3.2	1	30	0	0	yes	
	2	35	0	0	yes	
	3	57	0	1	yes	Tried to use the same name from the task before. That is not allowed and the dashboard alerted the user that the name introduced was already being used.
	4	44	0	1	yes	Tried to use the same name from the task before. That is not allowed and the dashboard alerted the user that the name introduced was already being used.
3.3	1	3	0	0	yes	
	2	2	0	0	yes	
	3	3	0	0	yes	
	4	2	0	0	yes	

Table 6.6: Usability question results

Question Id	User Id	Answer
1	1	correct
	2	correct
	3	correct
	4	correct
2	1	correct
	2	correct
	3	correct
	4	correct
3	1	incorrect
	2	incorrect
	3	incorrect
	4	incorrect
4	1	correct
	2	correct
	3	correct
	4	correct
5	1	correct
	2	correct
	3	correct
	4	correct
6	1	correct
	2	correct
	3	correct
	4	correct

The metrics to validate in these tests were:

1. Every user completes 80 % of the planned tasks.
2. There is a maximum of 3 errors in completed tasks;
3. Tasks do not take longer than the expected maximum duration;
4. 75 % of the users can correctly answer question 3;
5. 75 % of the users can correctly answer question 4;
6. 100 % of the users can correctly identify 6 out of the 8 parameters of question 6.
7. An average of 75 % positive answers in offline form's questions is received.

From the results on Table 6.5, it is visible that **requirements 1, 2, and 3 were satisfied** as every user performed every task within the maximum expected duration. Also, a maximum of 1 error occurred per task and when those errors occurred, the product prevented the continuation of the task and alerted the user of the problem. It was also seen that half of the users did not try to verify the existence of shortcuts in the functionality to plot resource metrics, even though they existed.

Observing Table 6.6, it is possible to say that **requirements 5 and 6 were also satisfied**, since every user responded correctly to questions 4 and 6. Regarding question 6, every user identified correctly all the parameters from the plot configuration file. Also, the other questions were all answered correctly, except for question 3 where any user could do it. Thereby **requirement 4 was not satisfied**. These results considering question 3 that the users could not understand when simulations can be applied, which is when a metric's values are being consulted in a chart, whether in a time interval or live, and there is data available. Even though the users were able to perform the task of going to the simulation page, they did not understand when they could do it. They just knew how to get to the simulations because a button was there. To provide better accessibility and clarity to this feature, a more direct option could be added to the menu. There, users would have to complete a form that would be verified after submission against the necessary restrictions to enter the simulation page.

Although not present in either of the results tables because the tasks were completed successfully, it was seen during the tests that users struggled a little at the first task when they had to choose between "Raw Data" and "Metric Data", and even some of them commented it during the test. So, another change that could be made is to change the naming of the "Raw Data" option to "Probed Data". Thus, it will be more clear that this option relates to data collected from probes while the other to data processed by TMA's Analyze component.

Concerning the last test metric to validate, the number 7, the results of the form's semantic differential and Likert scale questions are presented by Fig. 6.4 and Fig. 6.5 respectively. For semantic questions the answers "1" and "2" were considered positive, as for the Likert they were the "Agree" and "Partially agree" answers. Based on this, the % of positive answers for semantic differential and Likert questions was calculated and can be seen in Fig. 6.6 and Fig. 6.7 respectively. In summary, all questions, no matter the type, had at least 75 % of positive answers, except for the semantic differential question "How would you classify the clarity of when it is possible to apply simulations?" which only had 25 %. This exception reflects the problem mentioned above when requirement 4 ("75 % of the users can correctly answer question 3") was not satisfied. Performing the mean on the % of positive answers from all questions, a value of 82.5 % is achieved. Thereby, **requirement 7 was also satisfied**.

In the end, it could be said that the product is usable since 6 out of 7 requirements were satisfied. However, it presents a serious usability problem on the access to the feature of simulating metrics, which must be corrected.

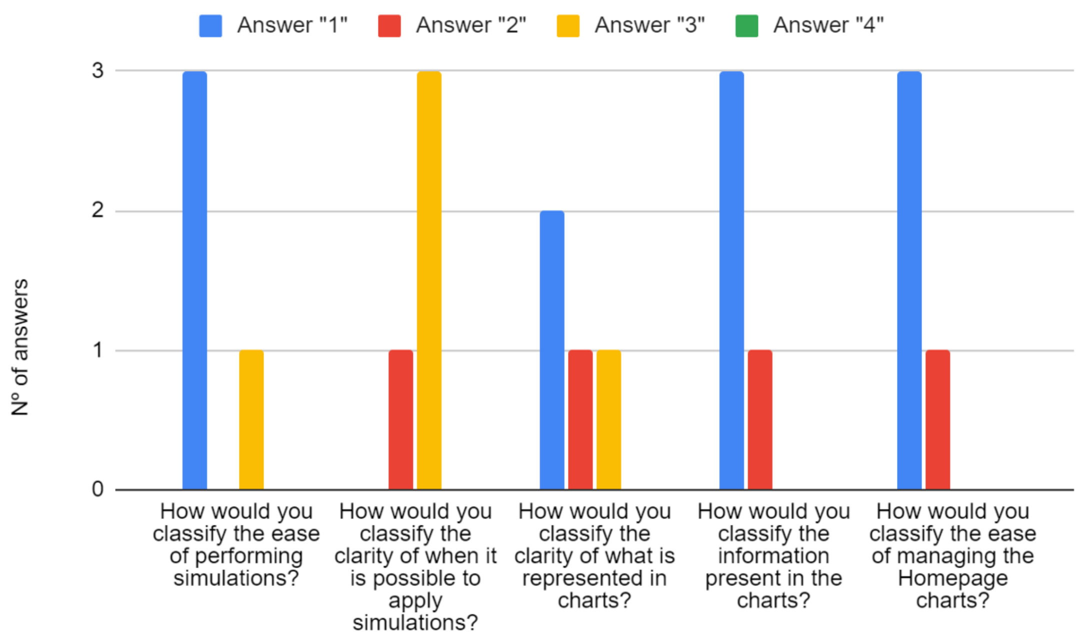


Figure 6.4: Semantic differential questions results.

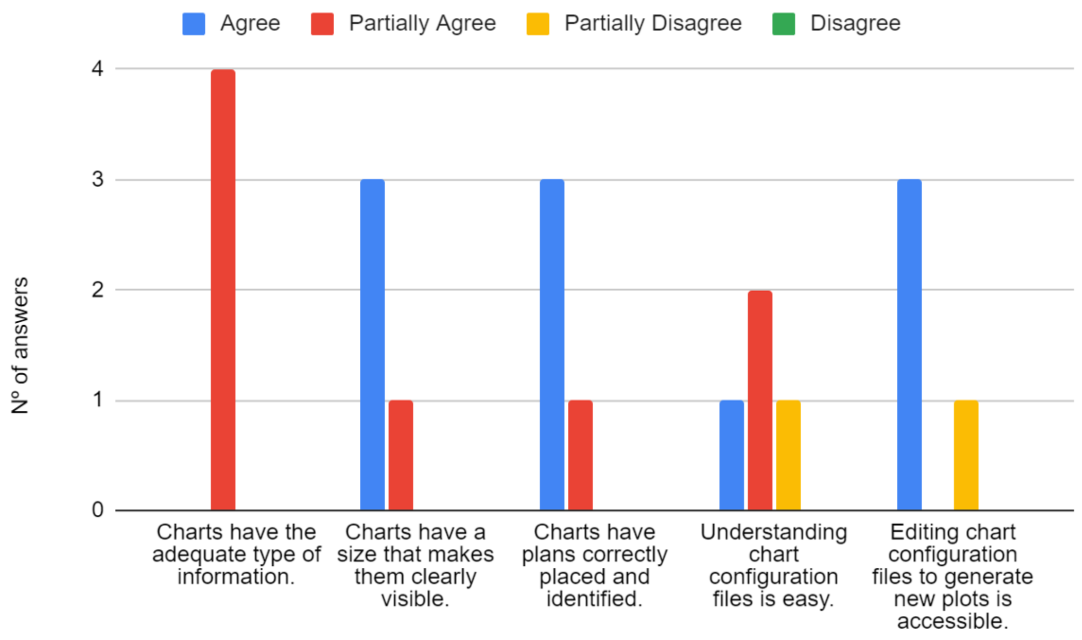


Figure 6.5: Likert questions results.

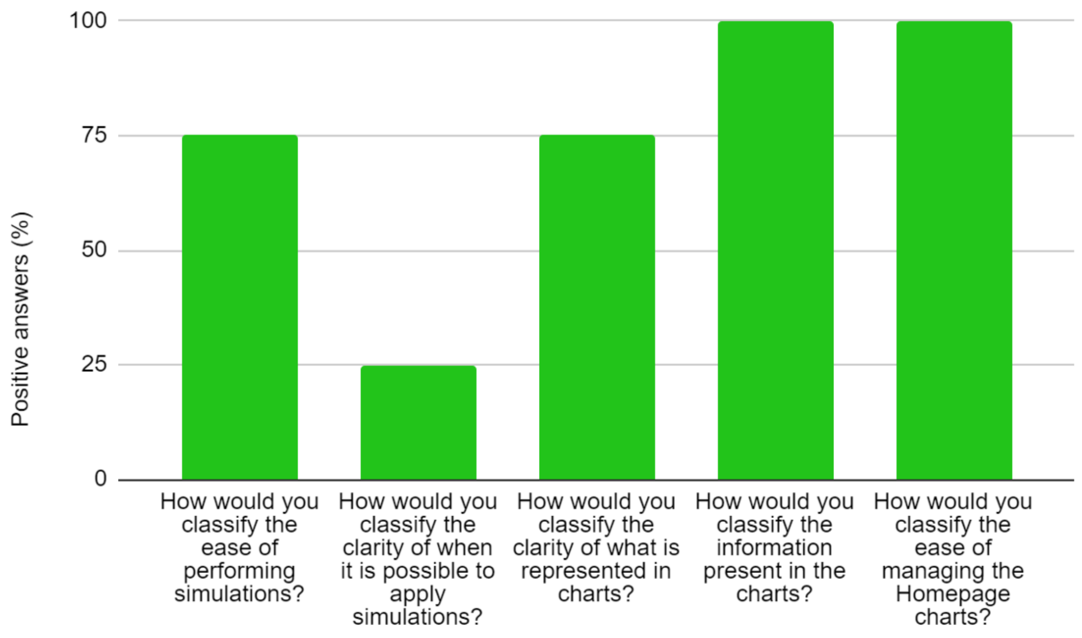


Figure 6.6: Semantic differential questions results as % of positive answers.

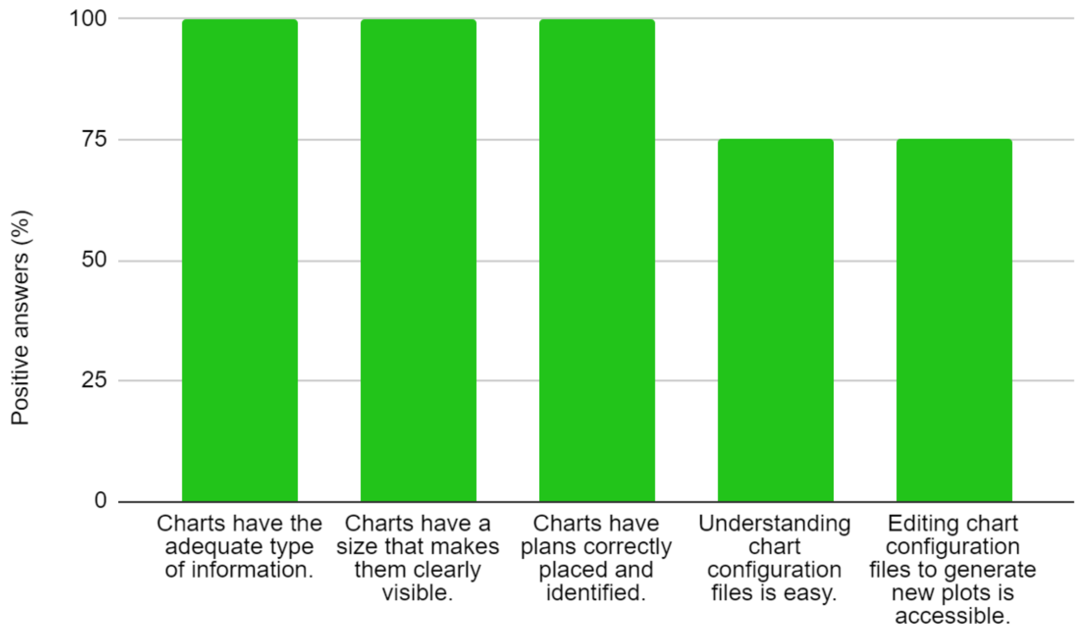


Figure 6.7: Likert questions results as % of positive answers.

6.4 Performance validation

This section presents and explains, primarily, the validation plan for the performance requirements presented in Section 4.2. Then, the results obtained are presented and analyzed.

6.4.1 Validation plan

The requirements elicited on performance concern mostly the dashboard, but also the performance of TMA's API. Those requirements are:

1. Having the dashboard's homepage full with live data plots, CPU consumption should not be higher than 1 virtual CPU and RAM consumption should not be higher than 250 MB;
2. The web page should not take longer than 3 seconds to be loaded;
3. A single kubernetes pod of the Application Programming Interface (API), consuming at maximum 1 virtual CPU, should be capable of handling, at minimum, 100 requests per second.

For each requirement, where it was necessary data collected from probes or data calculated by TMA's Analyze, an element was set to be monitored based on the quality model of the Fig. 6.8 and dummy rules that resulted in demonstration adaptations (prints in actuator's console) were used.

For the **first** requirement, the plan defined was:

1. Open Google Chrome browser with only 1 tab open;
2. In the tab open the dashboard's Single Page Application (SPA);
3. Navigate to the homepage in the SPA;
4. Add 6 different live plots;
5. Leave it running where each second, for each plot, requests will kept being made to update the charts values;
6. After an hour, using Chrome Task Manager tool, check consumption levels of CPU and RAM for the tab.

Concerning the **second** requirement, the plan defined was:

1. Host SPA web server;
2. Open ports to make the server accessible from any network;

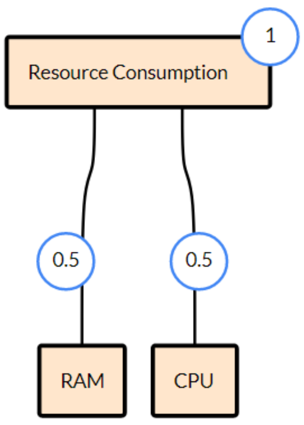


Figure 6.8: Quality model used for the performance validation.

3. Find out public Internet Protocol (IP) address;
4. Use Google's PageSpeed Insights (`https://pagespeed.web.dev/`) to analyze the load time.

For both the first and second requirements, the environment used had the following specifications:

- **CPU**: Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz 1.80 GHz
- **RAM**: 8 GB DDR4 2400 MHz
- **Disk**: Micron SSD 240 GB

Considering the **third** requirement, an experimental setup composed of 4 machines was used. One of them was used to generate load on the API, which is the machine used for requirements 1 and 2. The other 3 machines compose the Kubernetes cluster hosting TMA and their resources and main configurations are presented next:

- Control-plane (formerly known as master) machine with **4 vCPU**, **16 GB RAM** and **40 GB Disk**, that will be hosting an actuator as well;
- 2 worker machines with **4 vCPU**, **16 GB RAM** and **120 GB Disk**, where one of them will be hosting the API's pod limited to the maximum consumption of 1 CPU. The other worker will have the database pod and a probe sending data from another container present in the same environment to TMA.

The test environment of requirement 3 is presented in Fig. 6.9. First, TMA is configured to monitor and adapt an element (a pod's container running in the same machine as the probe), receiving data from a probe and executing adaptation plans based on dummy rules that will always be activated. For monitoring the element, the quality model from Fig. 6.8 is used. Thus, it becomes possible to

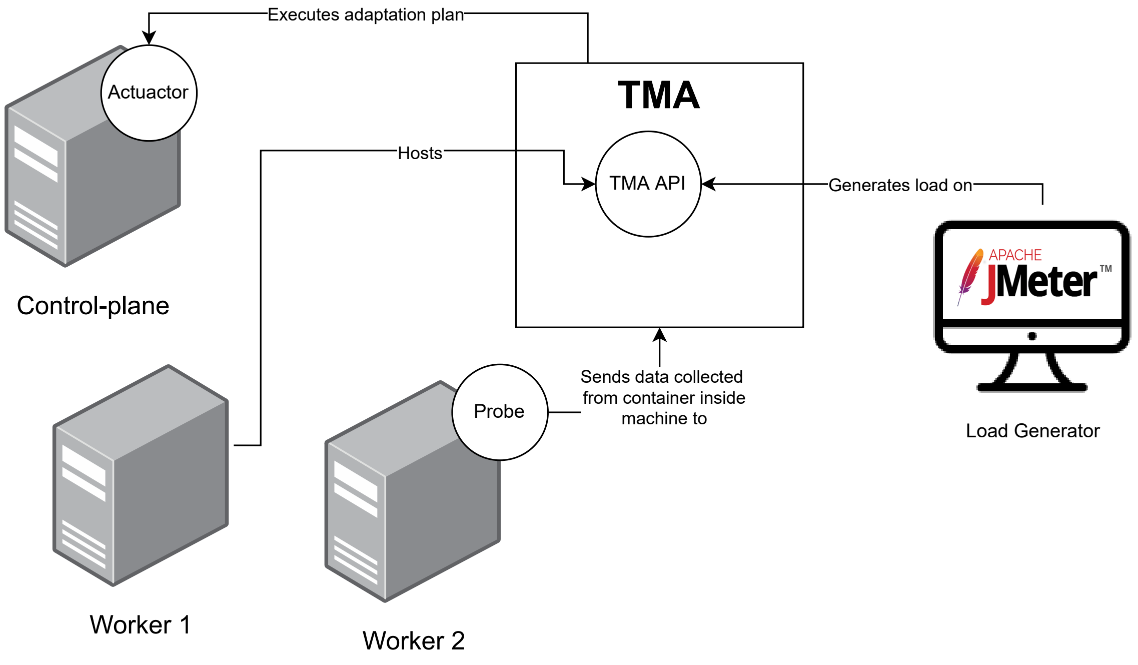


Figure 6.9: API performance test environment.

generate data necessary for performing some of the requests that define the load test.

The load is generated by the machine running JMeter (https://jmeter.apache.org/), and its demand is composed of 12 slots which are described by Table 6.7 and presented graphically in Fig. 6.10. Initially, there is a warmup period of 8 minutes where demand increases each slot (with a duration of 2 minutes) by 10 req/s, starting with 10 req/s. Then, the test itself starts with 50 req/s, increasing by 50 each slot (that lasts 3 minutes) until a final load of 400 req/s is reached. The length of the test is 32 minutes (1920 seconds) and it will be run 5 times.

Table 6.7: TMA's API performance test workload slot composition.

Slot	*Req/s*	*Duration(s)*
1	10	120
2	20	120
3	30	120
4	40	120
5	50	180
6	100	180
7	150	180
8	200	180
9	250	180
10	300	180
11	350	180
12	400	180

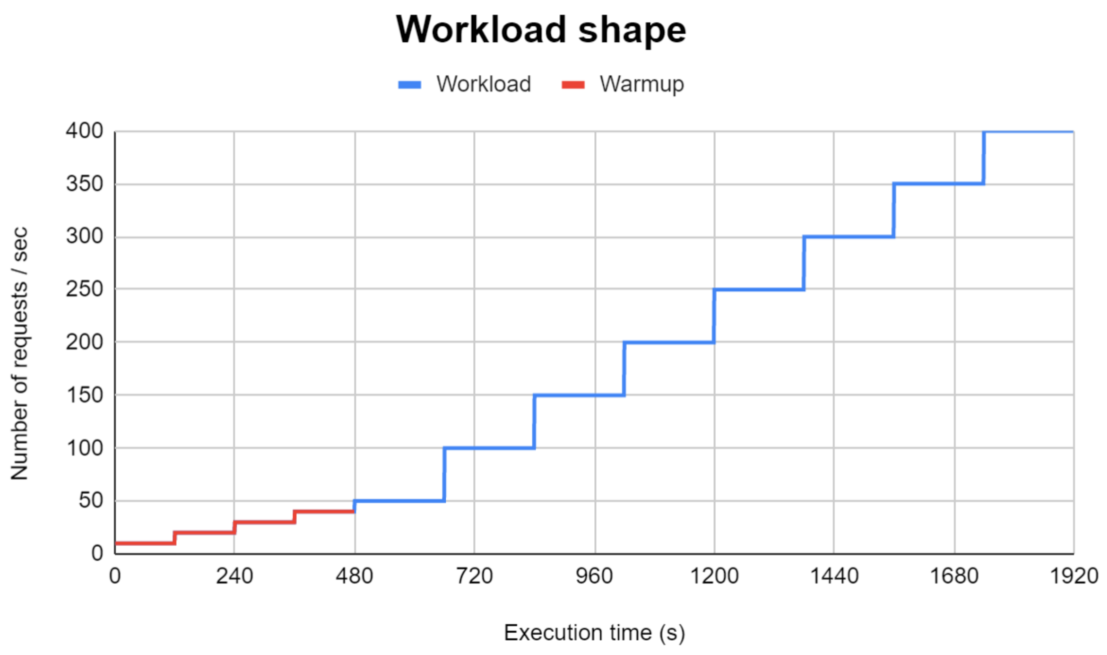


Figure 6.10: Workload applied in the API performance test.

As for the requests to perform on the test, they were selected based on the ones the dashboard performs when a user is using it. Also, the request set created intends to interact with every section of the dashboard. For instance, homepage chart management, visualization of adaptation rules, and plotting of data. For that, a set of 21 requests was created, where 4 of them update the database. The distribution of database reads and writes on the requests tried to follow the relation 80%/20%, respectively. It was defined like that since it is expected that the TMA's API will receive far more read requests than writes when the dashboard is being used. The sequential request execution is shown next:

1. Get homepage charts;
2. Get descriptions;
3. Create leaf metric;
4. Get metrics;
5. Get single parent metric;
6. Get single leaf metric;
7. Create parent metric;
8. Get quality models;
9. Get single quality model;
10. Get configuration profile;

11. Get metrics for quality model creation;
12. Create quality model;
13. Get active resources;
14. Get resource configuration profile;
15. Get configuration profile metrics list;
16. Get resource raw data;
17. Get resource metric data with plans;
18. Perform simulation on metric data;
19. Replace homepage chart;
20. Get rules;
21. Get rule code.

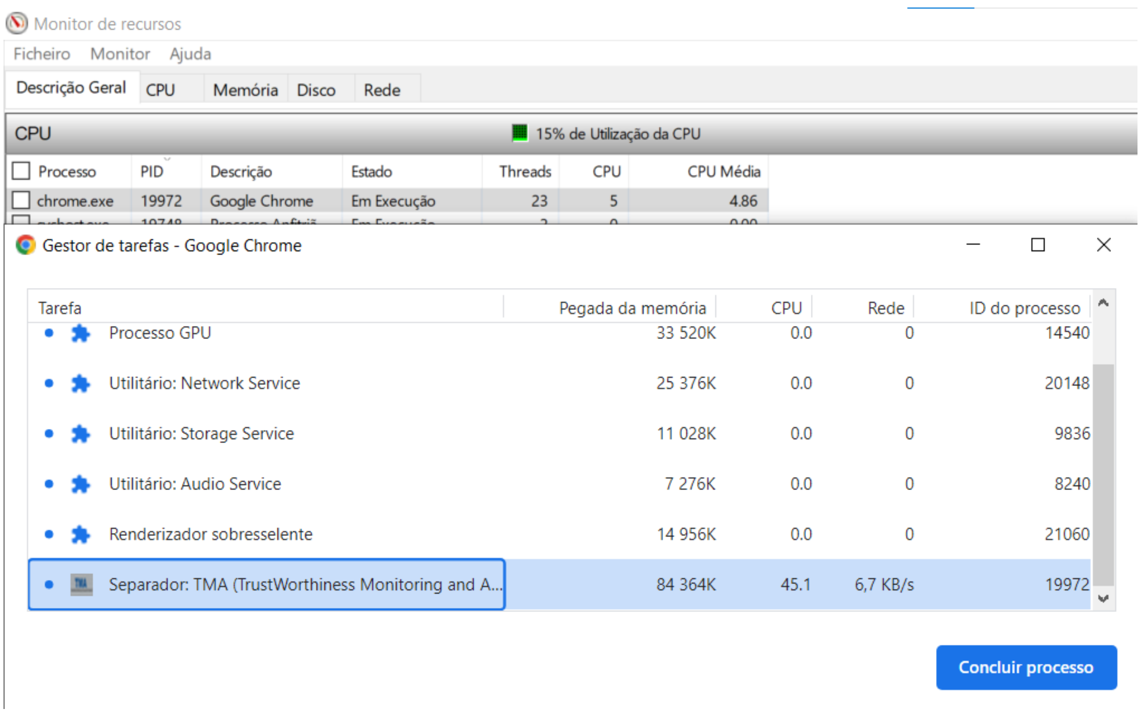
Some aspects must be noted regarding the execution of this request set. First, requests 7 and 12 are dependent on the success of request 3. So, if request 3 fails, they are not executed. Also, request 19 is set to be executed with a 10% probability. Besides, each request has a connect timeout of 300 milliseconds and a response timeout of 1 second. These values were defined based on the ones observed in the initial experiments when the TMA's API was not overloaded, and when the test request set was being built and JMeter features were being tested.

6.4.2 Results

The planned validation tests for performance were run and their results were obtained and processed. The requirements to validate were:

1. Having the dashboard's homepage full with live data plots, CPU consumption should not be higher than 1 virtual CPU and RAM consumption should not be higher than 250 MB;
2. The web page should not take longer than 3 seconds to be loaded;
3. A single kubernetes pod of the API, consuming at maximum 1 virtual CPU, should be capable of handling, at minimum, 100 requests per second.

To validate requirement 1, the homepage of the dashboard was left running for an hour, full of different live charts. During the experiment, Google Chrome's Task Manager tool was left open to monitor the CPU and RAM consumption values. Additionally to what was defined in the plan, the Window's Resource Monitor tool was used to get the mean CPU usage of the whole experiment run. The results, from both tools, are presented in Fig. 6.11. While Chrome's tool presents CPU usage considering 1 virtual CPU (which means values can be greater than 100%), the Window's tool presents the CPU usage considering the total number of virtual CPUs of the machine. As the CPU of the machine where the test was run has 4 cores, and 8 threads, the number of virtual CPUs is 8. That means that each machine's virtual CPU corresponds to 12.5% (100% ÷ 8 virtual CPUs) of the CPU. The "CPU Média" value from the figure is from Window's tool and it is the average CPU usage for the whole experiment, which has the value of 4.86, less than a half of 12.5. That means that the average usage of the dashboard tab was less than half of a virtual CPU. Besides, the value presented by Chrome's tool is showing the same results, 45.1 % of 1 virtual CPU. That is still less than half of a CPU. Also, from Fig. 6.11 it can be seen that the dashboard, at the end of the experiment, was consuming less than 100 MB. Thereby the **requirement 1 is satisfied**.



Monitor de recursos
Ficheiro Monitor Ajuda
Descrição Geral CPU Memória Disco Rede
CPU
🟩 15% de Utilização da CPU

Processo	PID	Descrição	Estado	Threads	CPU	CPU Média
chrome.exe	19972	Google Chrome	Em Execução	23	5	4.86

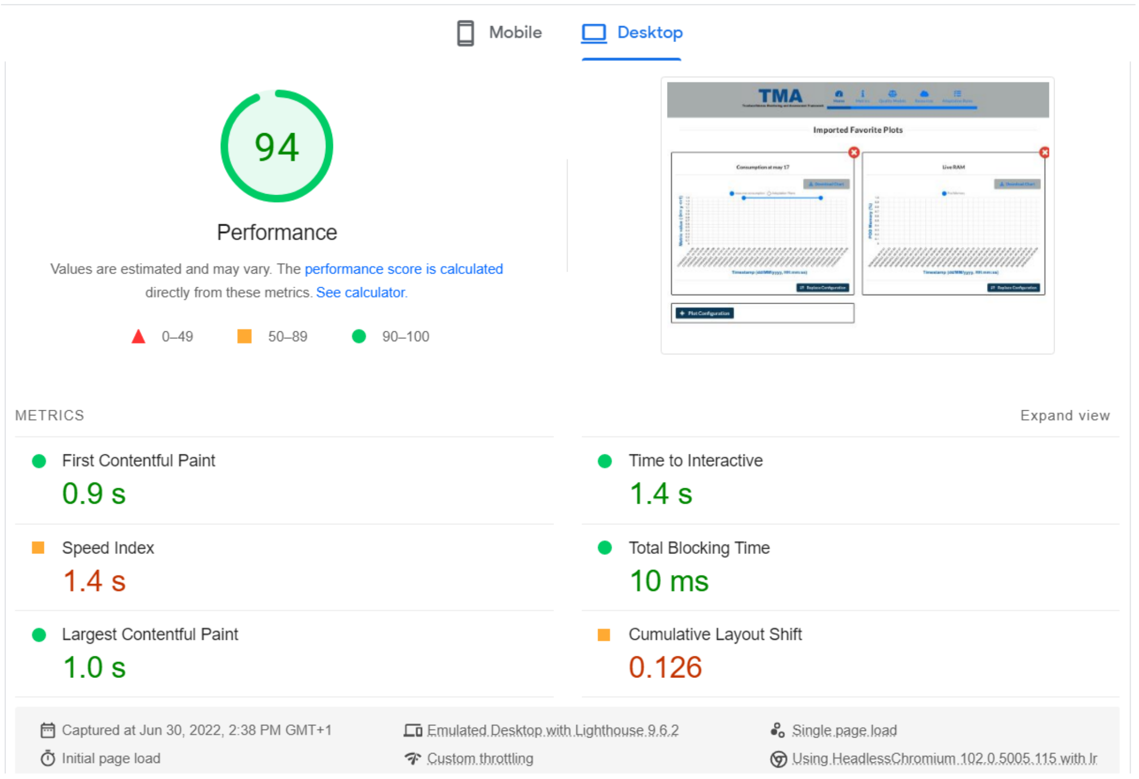
Gestor de tarefas - Google Chrome

Tarefa	Pegada da memória	CPU	Rede	ID do processo
Processo GPU	33 520K	0.0	0	14540
Utilitário: Network Service	25 376K	0.0	0	20148
Utilitário: Storage Service	11 028K	0.0	0	9836
Utilitário: Audio Service	7 276K	0.0	0	8240
Renderizador sobresselente	14 956K	0.0	0	21060
Separador: TMA (TrustWorthiness Monitoring and A...	84 364K	45.1	6,7 KB/s	19972

Figure 6.11: Dashboard's resource consumption after an hour.

Concerning requirement 2, after a server has been successfully set up for the dashboard and the traffic allowed, Google's PageSpeed Insights was used to analyze the page load time. Fig. 6.12 shows those results where a positive score was obtained, besides a page-load of 0.9 seconds for the "First Contentful Paint" metric and 1 second considering the "Largest Contentful Paint" metric. "Largest Contentful Paint" represents the time it takes for the largest text or image asset to load, while the "First Contentful Paint" is the same but applied to the first asset loaded [104]. Also, even if considering the "Time To Interactive" metric, which has the value of 1.4 seconds, the **requirement 2 is satisfied**.

Finally, for requirement 3, the throughput results of the load test are presented in Fig. 6.13. This figure presents the mean throughput considering 5 runs. As it can be observed, the TMA's API could handle until 200 req/s, but when the load in-



Mobile Desktop

94

Performance

Values are estimated and may vary. The performance score is calculated directly from these metrics. See calculator.

▲ 0–49 ■ 50–89 ● 90–100

METRICS Expand view

● First Contentful Paint 0.9 s	● Time to Interactive 1.4 s
■ Speed Index 1.4 s	● Total Blocking Time 10 ms
● Largest Contentful Paint 1.0 s	■ Cumulative Layout Shift 0.126

Captured at Jun 30, 2022, 2:38 PM GMT+1 · Emulated Desktop with Lighthouse 9.6.2 · Single page load

Initial page load · Custom throttling · Using HeadlessChromium 102.0.5005.115 with lr

Figure 6.12: Page load results of the dashboard, gotten from Google's PageSpeed Insights.

creased to 250 TMA's API pod reached its maximum CPU capacity. That resulted in the throttle of the service and the consequent performance degradation.

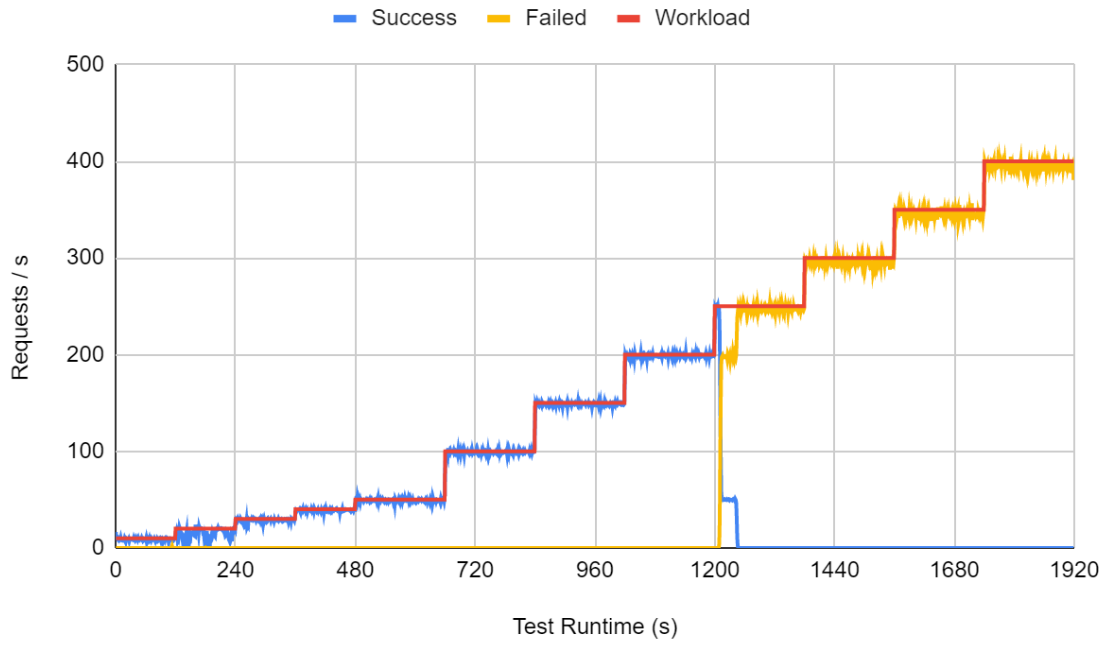


Figure 6.13: TMA's API mean throughput in performance test.

Additionally, to better analyze performance, and considering the 5 runs, metrics related to response time are presented in Table 6.8. This table considers the experiment from the beginning of the workload (without warm-up), at slot 5, until the end of slot 8 (480-1200s), and slot 9 (1200-1380s). As the table shows, and based on the mean and standard deviation, the response time between slots 5 to 8 is usually less than 100 ms, when the maximum reached within that interval was 653 ms. This means that in those slots performance was good as requests were being served in less than 700 ms and usually at times smaller than 100 ms. However, the same can not be said about Slot 9, which was when requests started to fail until the end of the experiment. Looking at the the response time results from Table 6.8, the mean value is 939.69. And this is a value very close to the timeout of 1 second set to the requests. The deviation and the fact of the value not being higher than 1000 is due to the successful requests at the beginning of the slot. To complement this analysis, Fig. 6.14 and Fig. 6.15 present the box plots for response time in slots 5-8 and slot 9. From these box plots, we can see that 75% of the response times are lower than a value smaller than 100 ms in slots 5-8. As for slot 9, the median value is slightly higher than 1000 ms and 50 % of the response times are above it. With that being said, **requirement 3 of performance is also satisfied**.

Table 6.8: Response time metrics, in ms, for Slots 5-8 and Slot 9.

Slot slice	*Mean*	*Standard deviation*	*Min*	*Max*
5-8	48.32	18.68	19	653
9	939.69	294.36	19	2040

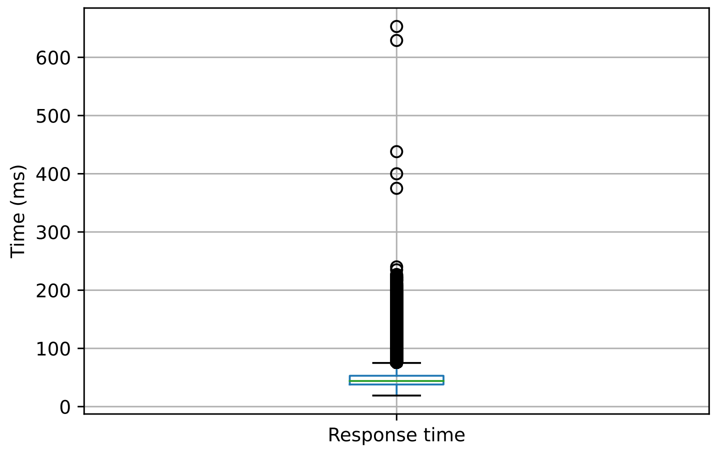


Figure 6.14: Slots 5-8 response time box plot.

In summary, all the performance requirements were fulfilled and therefore the product, in terms of performance, is valid.

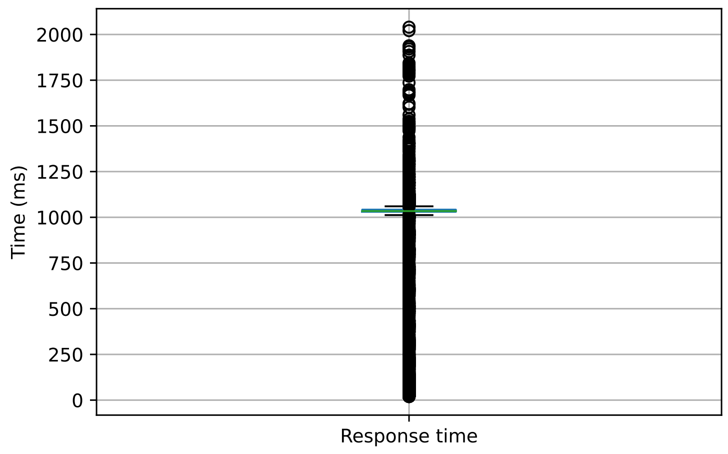


Figure 6.15: Slot 9 response time box plot.

6.5 Maintainability validation

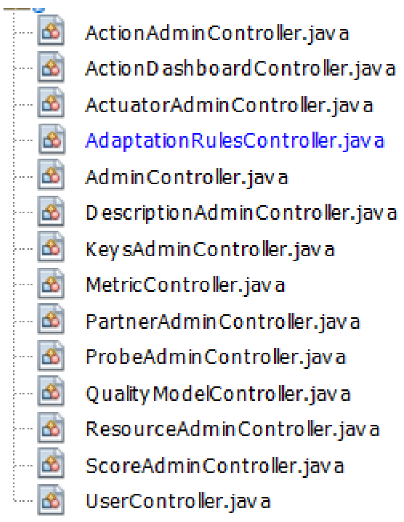
Maintainability was also one of the non-functional requirements defined for the product since there are plans for using TMA and the dashboard developed. However, this requirement was more related to programming methodologies to follow such as modularity, conventions, high cohesion, and loose coupling.

TMA and the created dashboard are intended to be changed, enhanced, and maybe restructured in the future. Additionally, errors may arise and the task of fixing them should be eased. For those reasons, facilitating the process of changes to the product was something to keep in mind when developing.

The following of this section will present maintainability aspects that were followed and used for developing the components of this dissertation.

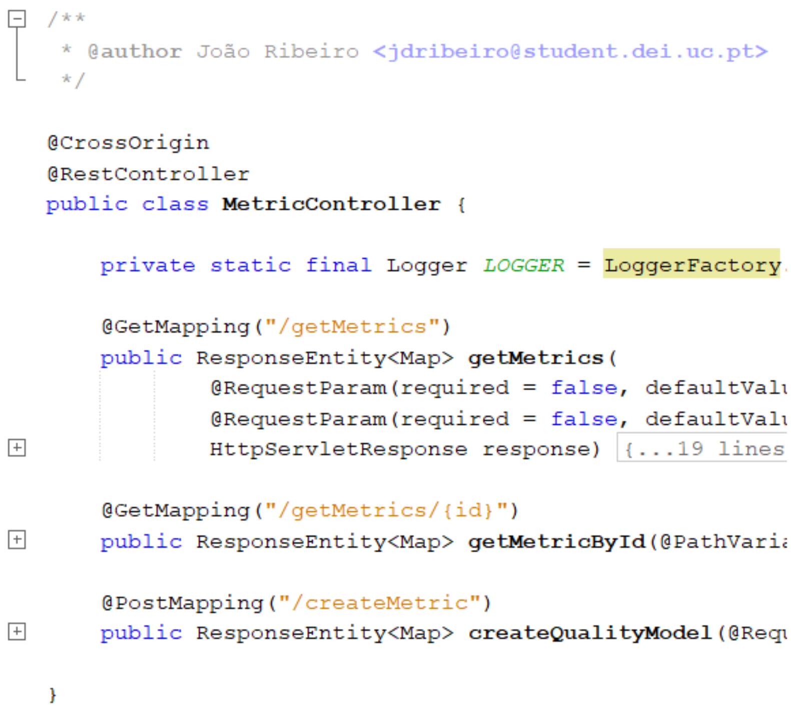
Considering *high cohesion and loose coupling*, there are some examples that followed that principle in the extension of TMA's API and the creation of the dashboard. Regarding the API, and as Fig. 6.16 shows, different controllers are holding a set of related endpoints. For example, the *MetricController.java*, from which the endpoints are presented in Fig. 6.17, is responsible for requests related to metrics, such as creation and listing. The same applies to the other controllers. Concerning the SPA implementation, the principle of loose coupling and high cohesion can also be seen in created components like DropDownDataFormat and ValidInputs, which were explained in Section 5.2.2. The first component holds a set of functions to convert different entities' data, received from the TMA's API, into a format acceptable by the DropDown component provided by Semantic UI. The ValidInputs component is composed of a set of functions that allows the validation of different types of inputs present in the interface.

With respect to conventions, those were followed in the Java programming of the TMA's API, Analyze and Planning components, as well as in the SPA project structure. The point of using conventions is to facilitate the comprehension of the



```
ActionAdminController.java
ActionDashboardController.java
ActuatorAdminController.java
AdaptationRulesController.java
AdminController.java
DescriptionAdminController.java
KeysAdminController.java
MetricController.java
PartnerAdminController.java
ProbeAdminController.java
QualityModelController.java
ResourceAdminController.java
ScoreAdminController.java
UserController.java
```

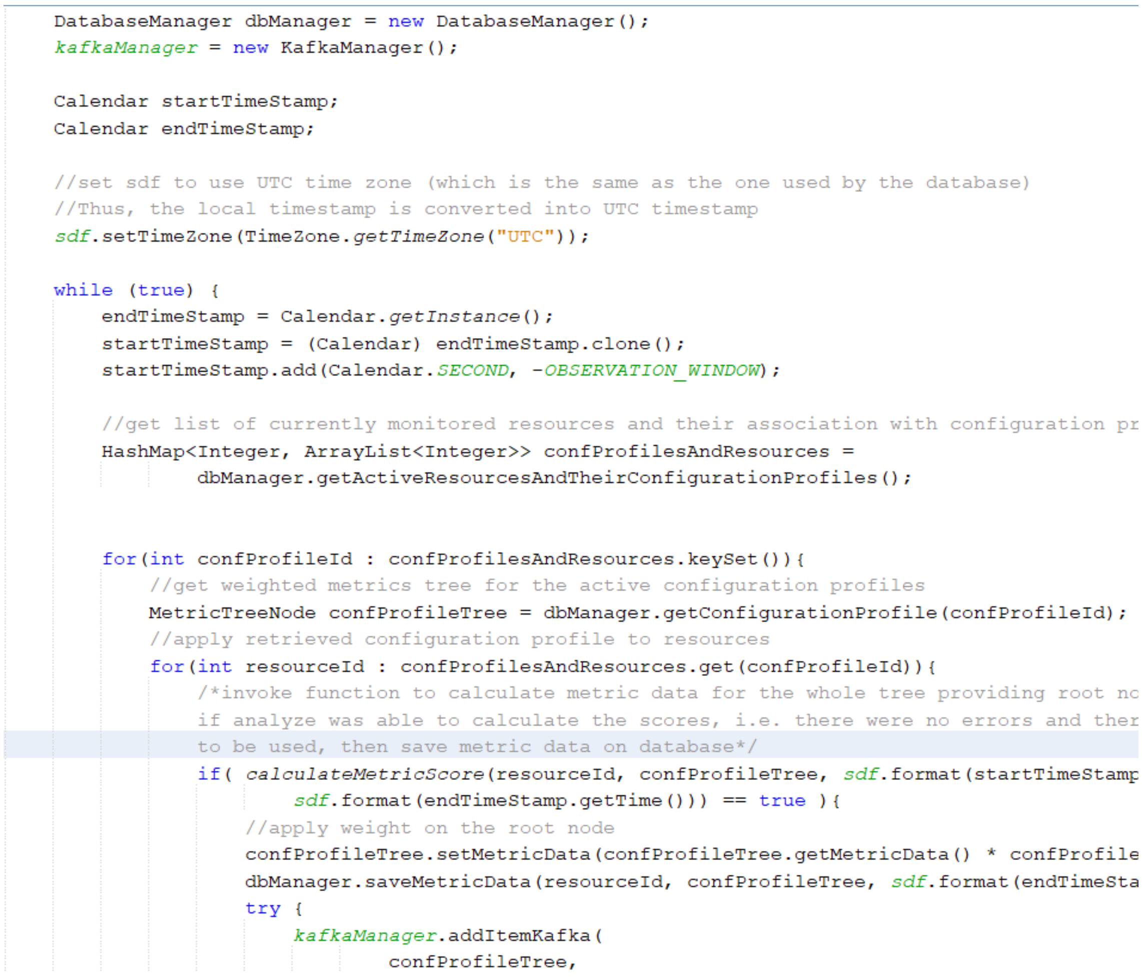
Figure 6.16: TMA's API implementation controllers.



```java
/**
 * @author João Ribeiro <jdribeiro@student.dei.uc.pt>
 */

@CrossOrigin
@RestController
public class MetricController {

    private static final Logger LOGGER = LoggerFactory.

    @GetMapping("/getMetrics")
    public ResponseEntity<Map> getMetrics(
            @RequestParam(required = false, defaultValu
            @RequestParam(required = false, defaultValu
            HttpServletResponse response) {...19 lines

    @GetMapping("/getMetrics/{id}")
    public ResponseEntity<Map> getMetricById(@PathVaria

    @PostMapping("/createMetric")
    public ResponseEntity<Map> createQualityModel(@Requ

}
```

Figure 6.17: TMA's API MetricController.java endpoints.

code, and the type of components involved. For instance, in Java there are naming conventions for classes, interfaces, variables, methods, and so on. The Java code implemented followed naming and indentation conventions and Fig. 6.18 presents just an example. Regarding the SPA, it is written in React, and React applications have well-structured and organized projects, usually having 2 common folders, components, and pages. Fig.6.19 demonstrates that a well-defined organization and structure were used for the dashboard's project. Created components are inside the "components" folder, while each navigable page is placed under the "pages" folder. The "semantic-ui" folder contains Semantic UI theming properties that had to be defined to provide a coherent and consistent design across the dashboard. The "configurations" folder holds the information regarding the path of TMA's API. Finally, "utils" folder contains components that become useful in multiple parts (components and pages) of the dashboard. This folder includes, for example, the *APIModule*, *ValidInputs* and *DropDownDataFormat* components.



```
DatabaseManager dbManager = new DatabaseManager();
kafkaManager = new KafkaManager();

Calendar startTimeStamp;
Calendar endTimeStamp;

//set sdf to use UTC time zone (which is the same as the one used by the database)
//Thus, the local timestamp is converted into UTC timestamp
sdf.setTimeZone(TimeZone.getTimeZone("UTC"));

while (true) {
    endTimeStamp = Calendar.getInstance();
    startTimeStamp = (Calendar) endTimeStamp.clone();
    startTimeStamp.add(Calendar.SECOND, -OBSERVATION_WINDOW);

    //get list of currently monitored resources and their association with configuration pr
    HashMap<Integer, ArrayList<Integer>> confProfilesAndResources =
            dbManager.getActiveResourcesAndTheirConfigurationProfiles();


    for(int confProfileId : confProfilesAndResources.keySet()){
        //get weighted metrics tree for the active configuration profiles
        MetricTreeNode confProfileTree = dbManager.getConfigurationProfile(confProfileId);
        //apply retrieved configuration profile to resources
        for(int resourceId : confProfilesAndResources.get(confProfileId)){
            /*invoke function to calculate metric data for the whole tree providing root nc
            if analyze was able to calculate the scores, i.e. there were no errors and ther
            to be used, then save metric data on database*/
            if( calculateMetricScore(resourceId, confProfileTree, sdf.format(startTimeStamp
                    sdf.format(endTimeStamp.getTime())) == true ){
                //apply weight on the root node
                confProfileTree.setMetricData(confProfileTree.getMetricData() * confProfile
                dbManager.saveMetricData(resourceId, confProfileTree, sdf.format(endTimeSta
                try {
                    kafkaManager.addItemKafka(
                            confProfileTree,
```

Figure 6.18: Followed Java's naming and indentation conventions example.

Finally, the modularity principle was followed in the implementation of TMA's Analyze processing methods. In short, for leaf metrics, Analyze uses methods for normalizing data collected from probes and aggregating it. Also, it uses other methods for calculating the values for parent metrics from their children. There are some methods implemented however, a need to add new methods and types

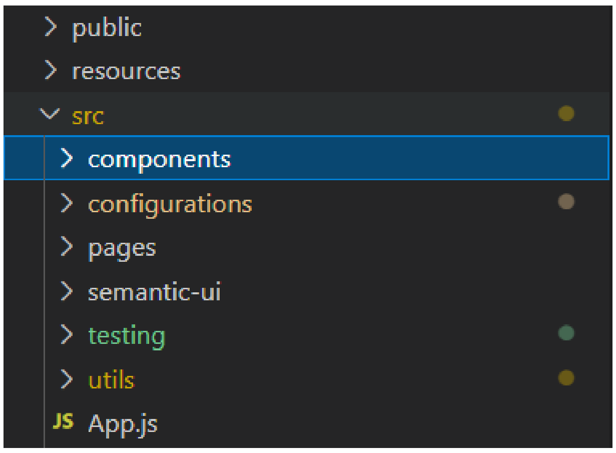
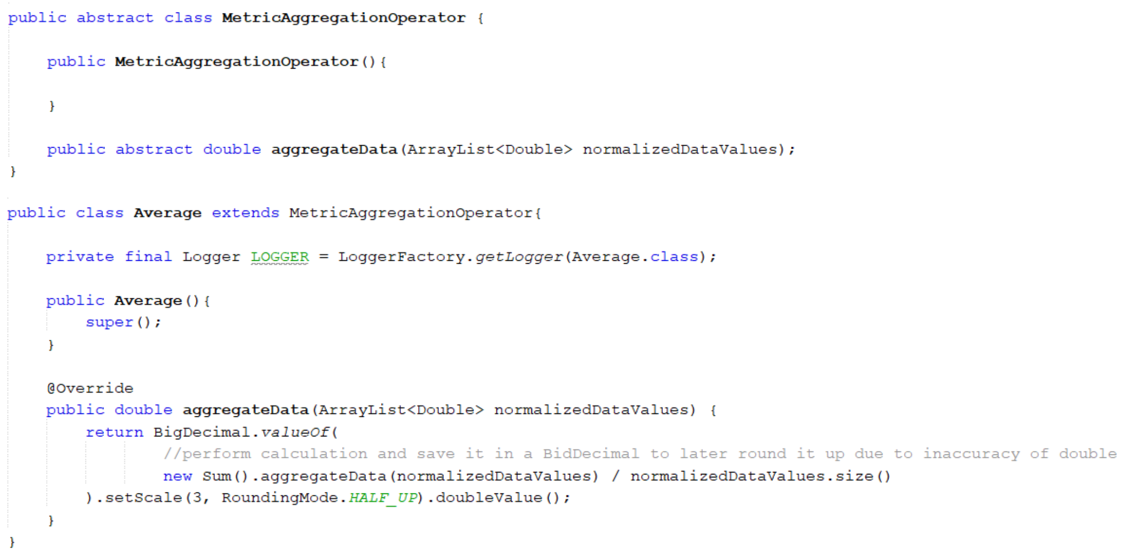


Figure 6.19: SPA's React project structure.

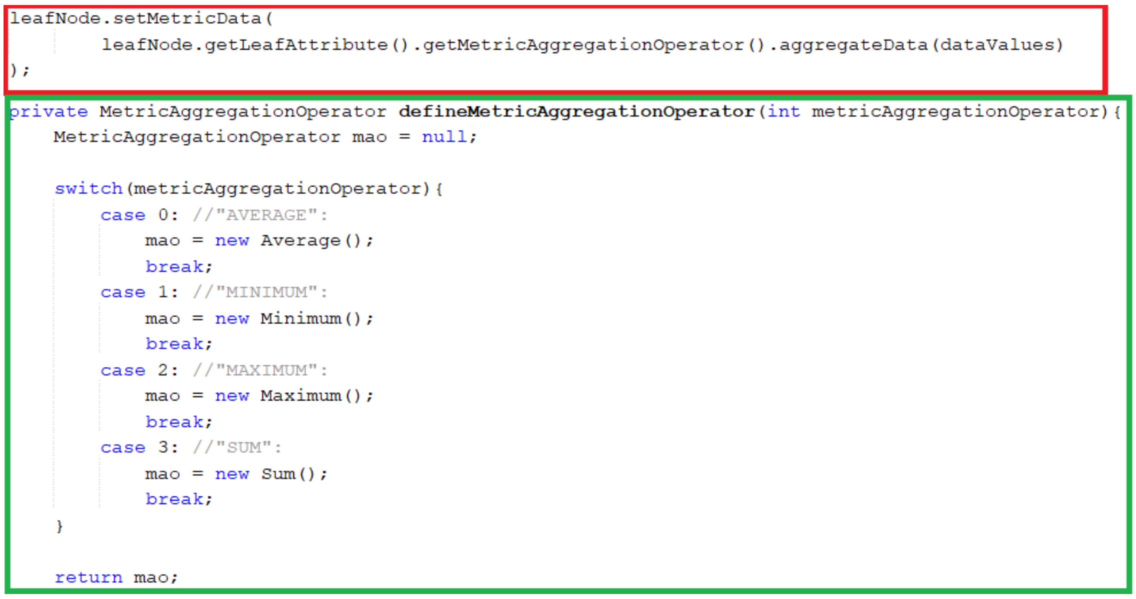
of aggregations may arise in the future. Therefore, to allow those additions, Analyze makes use of abstract classes. When it reads the quality models from the database, Analyze associates the abstract class variable with an instance of a class that implements the abstract method. Thus, Analyze's logic code does not need to be changed, because it always invokes the same method. Instead, a class extending the abstract is the one responsible for implementing the method's logic. In Fig. 6.20 the declaration of the abstract class used for the leaf metrics aggregation operator is shown, along with a concrete implementation for it, the average operator. In Fig. 6.21, a piece of the Analyze's code invoking the aggregation method from a quality model's leaf node is shown in red, as in green it is presented the initialization of that method in the quality model.



```
public abstract class MetricAggregationOperator {

    public MetricAggregationOperator(){

    }

    public abstract double aggregateData(ArrayList<Double> normalizedDataValues);
}

public class Average extends MetricAggregationOperator{

    private final Logger LOGGER = LoggerFactory.getLogger(Average.class);

    public Average(){
        super();
    }

    @Override
    public double aggregateData(ArrayList<Double> normalizedDataValues) {
        return BigDecimal.valueOf(
                //perform calculation and save it in a BidDecimal to later round it up due to inaccuracy of double
                new Sum().aggregateData(normalizedDataValues) / normalizedDataValues.size()
        ).setScale(3, RoundingMode.HALF_UP).doubleValue();
    }
}
```

Figure 6.20: Aggregation method abstract class and an implementation example.

In the end, maintainability practices were followed to ensure that the understanding of the code implemented is facilitated and that any necessary future changes can be performed at minimal cost and effort.



```
leafNode.setMetricData(
        leafNode.getLeafAttribute().getMetricAggregationOperator().aggregateData(dataValues)
);
```

```
private MetricAggregationOperator defineMetricAggregationOperator(int metricAggregationOperator){
    MetricAggregationOperator mao = null;

    switch(metricAggregationOperator){
        case 0: //"AVERAGE":
            mao = new Average();
            break;
        case 1: //"MINIMUM":
            mao = new Minimum();
            break;
        case 2: //"MAXIMUM":
            mao = new Maximum();
            break;
        case 3: //"SUM":
            mao = new Sum();
            break;
    }

    return mao;
```

Figure 6.21: Analyze's invocation of aggregation method, in red, and operator's initialization in the quality model, in green.

Chapter 7

Management

In this chapter activities and concepts related to the management activity of the product developed are presented. The topics going to be presented are development methodology, requirements prioritization and management, tasks scheduling, and risk analysis.

7.1 Development model

In this dissertation *Waterfall* was selected as the development model to follow.

Waterfall is a sequential software development model, where progress flows steadily through the phases of a project towards the conclusion. Moreover, it involves documenting a project in advance, including all the features, variations, and outcomes [105].

The goal of using this methodology is to conduct a controlled process where steps of the methodology are concluded, and from there and beyond not revisited anymore. The idea is to collect all the necessary knowledge in advance and from there precisely plan and estimate the tasks until completion. That is why a lot of effort was spent on detailing the requirements.

The sequential phases in the Waterfall model, presented by Fig. 7.1, are [106]:

- **Requirements Analysis**: Here, all possible requirements of the system to be developed are captured and an analysis of those that should be implemented is done;
- **System Design**: This is the phase where the architecture for the product is studied and created based on the requirements;
- **Implementation**: Step of the methodology where the system is built according to the architecture and requirements;
- **Testing**: At this phase, all the units developed in the implementation phase are tested for detecting errors;

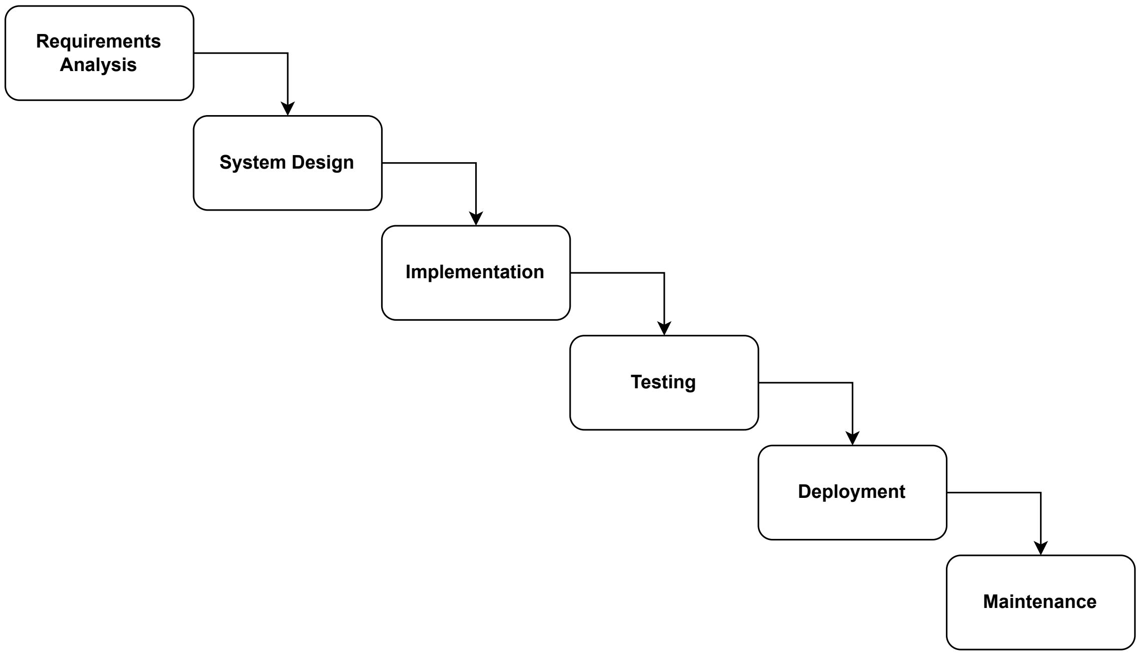


Figure 7.1: Waterfall model steps (adapted from [106]).

- **Deployment**: Once the testing phase is finished successfully, the product is deployed in an environment or released into the market;
- **Maintenance**: During production, issues might come up and they need to be fixed. In this phase, those identified issues are fixed, and continuous improvement of the product is performed.

7.2 Requirements management

Along with the use cases approach for eliciting requirements, the *MoSCoW* method was used to rank the use cases and establish development priorities among them. As the descriptions, the prioritization of the use cases can be consulted in Appendix A.

The goal of prioritizing is to understand the most critical requirements, in what order to develop them, and what not to deliver if there is pressure on resources [107]. To distribute priorities across the use cases, this method uses 4 types of priority which have the following meanings according to [108]:

- **Must have:** represent non-negotiable needs for the project, product, or release in question. If the product won't work without that use case or the release becomes useless without it, then the functionality is a "must-have";
- **Should have:** are essential needs to the product but not vital as the must requirements. If they are left out, the product still functions, however, they may add significant value;

- **Could have:** another way of describing this category is nice-to-haves. This type of priority means that the requirements are not necessary to the core function of the product. When compared with "should-have", they have a much smaller impact on the outcome if left out;

- **Will not have:** this priority allows to define functionalities that will not be included in a specific release of a product, thus the team knows they are not a priority for this specific time frame. They also help control the scope of the project once requirements implementation negotiations are made upon the *MoSCoW* prioritization method.

A few weeks after the start of the development of the dashboard, important (Must have) requirements were noticed to be missing. The requirements not foreseen were related to the management of charts on a homepage. Thus, requirements with the ids *4.10*, *11.1*, *11.2*, and *11.3* were elicited at this phase of the project and were assigned the *Must* priority. Consequently, that led to a rearrangement of some of the other requirements priorities. The list of occurred transitions in the priority of requirements is presented next:

- **Must to Should**: Requirements *9.1*, *9.2*, *10.1*, and *10.2*;

- **Should to Could**: Requirements *3.1*, *4.1*, *5.1*, *6.1*, and *7.1*;

Fig. 7.2 presents a summary of the number of requirements elicited within each priority, before and after the addition of the new requirements. Also, it is shown the number of implemented requirements for each priority.

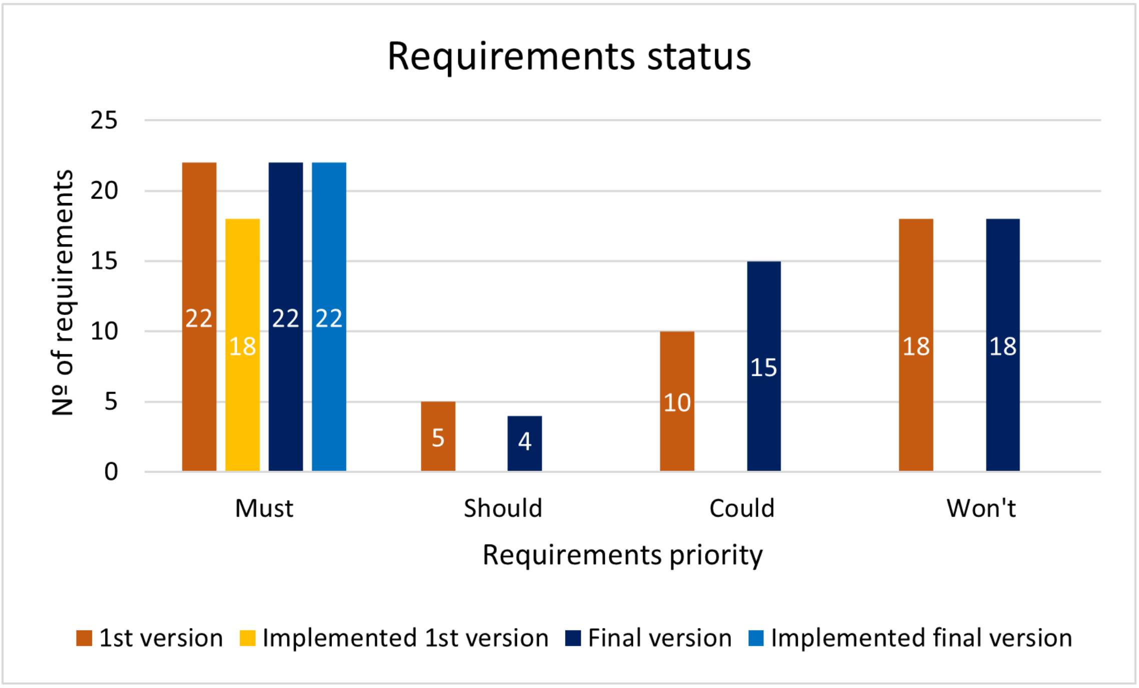


Figure 7.2: Elicited and implemented requirements considering the versions before and after the addition of new requirements.

7.3 Tasks scheduling

For planning the length of activities Gantt diagrams were used. The diagram presented on Fig. 7.3 contains the planned length of activities for the first semester.

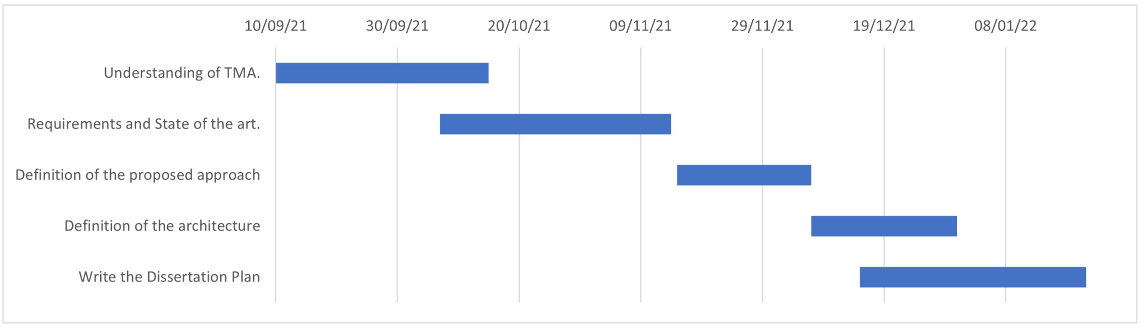


Figure 7.3: Planned Gantt diagram for the first semester.

However, during this semester the requirements and state-of-the-art analysis took more time than expected. For that reason, the tasks "Definition of the proposed approach" and "Definition of the architecture" had a shorter time to be completed. Also, the start of the activity "Write the Dissertation Plan" had to be started sooner due to incompatibilities with other projects on the planned start date. Fig. 7.4 presents the actual start time and length of the defined tasks in the first semester.

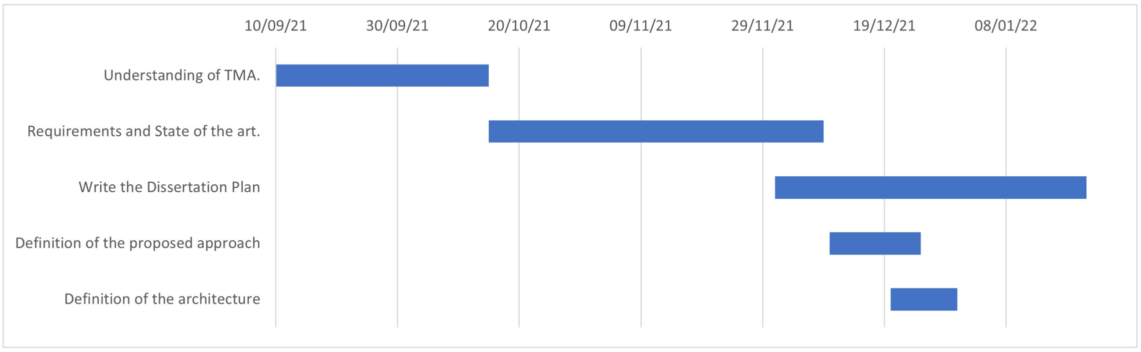


Figure 7.4: Actual Gantt diagram for the first semester.

The planned activities for the second semester are presented in Fig. 7.5. The idea was to initially adjust some aspects concerning non-functional requirements and architecture, using the feedback received from the middle-term presentation. Then, it would be configured a dummy React App so that a communication basis between the dashboard and TMA-API could be established. From there on, the development would start with the creation and listing of the quality model's related entities, as well as with the reprogramming of Trustworthiness Monitoring & Assessment Framework (TMA)'s Analyze. Then, functionalities related to plotting and simulating metrics would be developed. At that point, the main features related to the monitoring of systems would be completed and, thereby, a first validation phase would be performed on what was implemented. Next, the functionalities concerning the management of adaptation rules would be created, followed by the implementation of listing and consulting of adaptation plans and logs. Also, the listing and consulting of details from other TMA's entities (Description, Resource, Probe, Action, and Actuator) would be developed next. Finally, a validation of the functionalities implemented after the first validation phase would be performed, where the writing of a paper should be started and meanwhile the writing of the dissertation document.

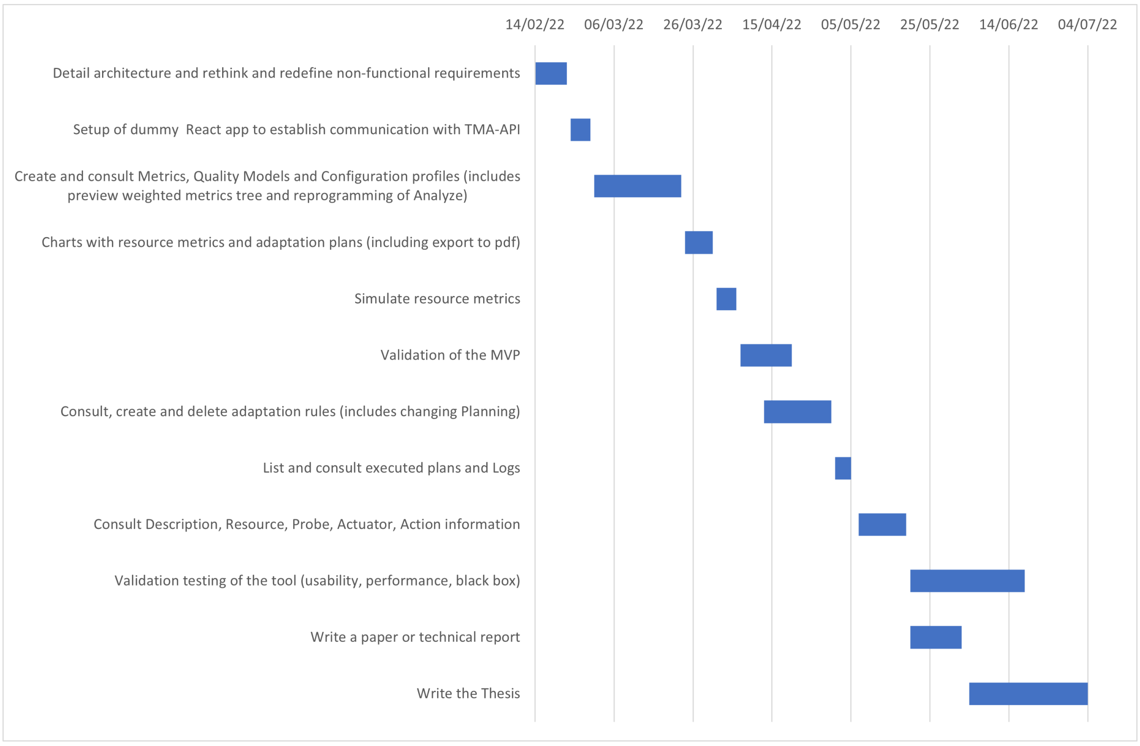


Figure 7.5: Planned Gantt diagram for the second semester.

However, some adjustments had to be made to the plan when requirements changed. Those changes are reflected in Fig. 7.6. In short, the new elicited requirements' implementation was added before the first validation phase, and the task of implementing the listing and consulting of details of TMA's entities (Description, Resource, Probe, Action, and Actuator) was excluded.

Even with the adjustments made when requirements changed, the plan could not be followed entirely. The actual activities performed and their duration is presented in Fig. 7.7. Basically, the first validation phase planned was not performed, since some of the previous tasks took more than expected to implement, such as the preview of the weighted metrics tree and the proper generation of charts and their exportation. Also, the task of implementing the listing and consulting of details from adaptation plans and logs could not be performed. That happened because previous tasks, such as the management of homepage charts, took more than expected, and an additional task was performed to give a branding design to the dashboard. Finally, and during the development phase, some errors had to be corrected, but this activity extended a little further to the end of the implementation phase, and that delayed the start of the following activities.

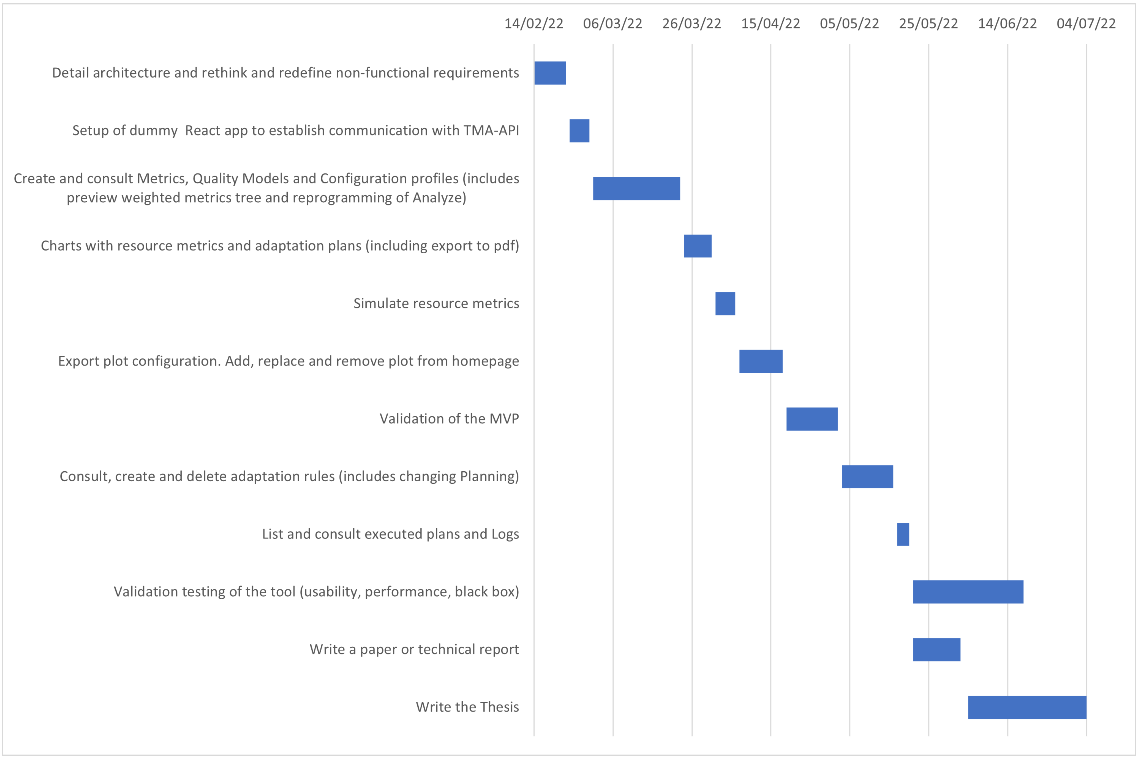


Figure 7.6: Planned Gantt diagram for the second semester after requirements changed.

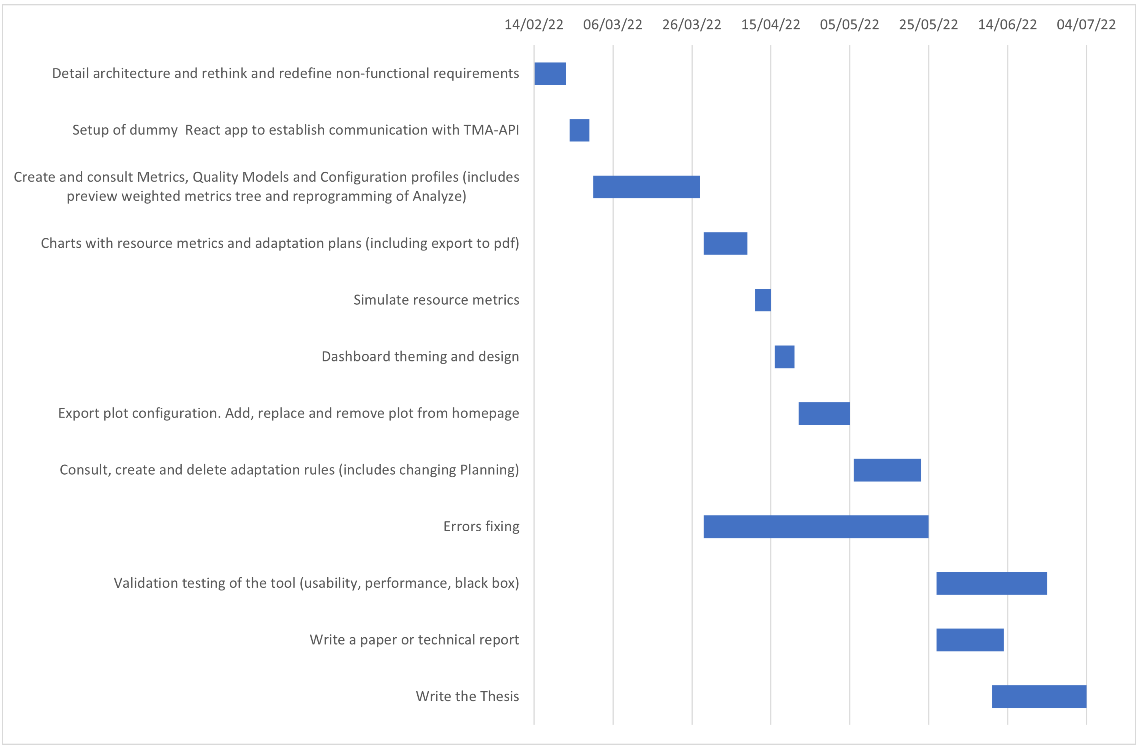


Figure 7.7: Actual Gantt diagram for the second semester.

7.4 Risk analysis

In the management of this product, a risk analysis was conducted based on the risk management framework proposed by Christopher Alberts and Audrey Dorofee [109]. The methods presented for risk statement, identification, and mitigation were used to draw the risks of this dissertation and plans to mitigate them. Like the framework, continuous risk assessment was performed. Thus, in the first semester, the first draft of risks was created, and they were revisited and updated monthly.

Table 7.1 shows, for each identified risk, an id, a condition, a consequence, a mitigation plan, and a date of identification.

Table 7.1: Risks Analysis.

Id	**Condition**	**Consequence**	**Mitigation Plan**	**Date of Identification**
R-1	Functional requirements elicited need new features on TMA's Analyze and Planning core components.	May lead to more time than estimated to implement those features.	Firstly implement the features concerning the Analyze, as they contribute most to product value.	15/10/ 2021
R-2	TMA was created some years ago and has not been maintained and updated.	Might lead to the presence of unknown bugs which causes delays in the GUI development.	Found bugs should not take more than 1 day to fix. If they do, begin to develop other GUI functionalities that do not involve the malfunction.	15/10/ 2021
R-3	I have never developed a GUI unless for academic purposes with basic functionalities.	May lead to less quality of the product.	Research for frameworks that may ease building the GUI. Compare their suitability for the project and compatibility.	15/10/ 2021

Id	Condition	Consequence	Mitigation Plan	Date of Identification
R-4	I have little or no experience with frontend frameworks.	May lead to extra time in development due to learning purposes.	Look for tutorials that teach the concepts and usage of the chosen frameworks.	15/10/ 2021
R-5	TMA components' Dockerfiles do not specify the versions of the technologies used at the time they were created.	Might lead to errors of currently unsupported features, delaying the development as compatible technologies versions have to be found.	Deploy TMA before developing starts, and if errors come up due to deprecated features, look for the technology version that supports them and specify it in the Dockerfile.	15/11/ 2021
R-6	The dashboard implementation requires the use of other libraries.	May lead to integration problems, delaying development.	Try to use well-documented, exemplified, and popular libraries.	15/02/ 2022
R-7	The performance validation will use JMeter, which I never used, to generate load on TMA's API.	May lead to delays in setting up and, consequently, executing the validation task.	Look for tutorials to ease the task of building the load script for JMeter.	15/02/ 2022
R-8	New requirements have been elicited and they are a priority.	May lead to the impossibility of implementing some planned requirements.	Perform estimation for new requirements and, if necessary, negotiate and exclude requirements not so important from the plan.	15/03/ 2022

Id	Condition	Consequence	Mitigation Plan	Date of Identification
R-9	Requirements are taking longer to implement than expected.	May lead to delay of the following planned tasks.	Exclude the task of performing a first product validation from the plan.	15/04/2022
R-10	Performance validation test is taking longer than expected to set up.	May lead to unfinished and incomplete tests at the planned end for the validation task.	Stop doing parallel validations on the product, and stop writing the dissertation. Focus on figuring out the problems and setting up the performance test environment.	15/05/2022

From the risks presented all have been mitigated.

Risk *R-1* was mitigated when the development ended since at that time both TMA's Analyze and Planning were already reprogrammed.

Risk *R-2* stopped being a risk when maintenance had to be performed on TMA and it was integrated with success in the TalkConnect project. The success of the project meant that TMA was validated.

Risk *R-3* has been mitigated while performing this document where analysis of frontend frameworks has been made.

As for *R-4*, since the identification of the risk, the framework chosen to implement the product of this dissertation, React, was used in another project in the first semester. So, its concepts have already been learned. In that same recent project, a CSS framework was also used. Thereby, using Semantic UI (which will be integrated with React), should not be hard to learn and use, once its documentation structure is similar to the one from the other CSS framework.

Respecting *R-5*, the mitigation plan was performed and issues were found and corrected.

Concerning *R-6*, the mitigation plan was followed. However, delays due to the integration of libraries were still a common existence.

Regarding risks *R-7*, *R-8*, *R-9* they were mitigated following the plans. *R-8* is related to the requirements priority change when new, and more important, requirements were elicited. As mentioned before in this chapter, that resulted in the exclusion of the implementation of listing and consulting of details from TMA's Description, Resource, Probe, Action, and Actuator entities. As for *R-9*, it was still a risk for a while, where its mitigation plan was applied twice. First, when the first validation phase was removed from the plan, and, secondly when the listing and consulting of details from adaptation plans and logs were removed from the plan.

With respect to *R-10*, it was mitigated when the plan was applied, but that could not be the case. The situation was that multiple tasks were being performed in parallel and each time a performance test was left running, errors of various kinds were showing up. Thus, the mitigation plan is just defined to set the focus on executing the performance tests successfully.

Chapter 8

Conclusion and Future Work

In the past few years, Cloud usage has increased largely. And under such a complex environment as the Cloud, monitoring systems become an important activity to perform since applications are exposed to multiple events that may cause disruptions to services.

More than just monitoring, if self-adaptive capabilities can be implemented on systems, the amount of work to be performed by Information Technology (IT) supervisors can be reduced. This happens as manual adaptations stop being required for systems to adapt. Thus, professionals' time can be spent on other tasks.

Nonetheless, either in monitoring or in configuring self-adaption actions for systems, it is important to have tools that can support the analysis of collected system data and provide insights on how to better configure adaptations.

The project of this dissertation had the goal of providing all the mentioned aspects through a dashboard for decision support, served as a Single Page Application (SPA). This dashboard is built for integration with Trustworthiness Monitoring & Assessment Framework (TMA), a framework that allows self-adaptiveness capabilities on systems. The support tool, meant to integrate with TMA, provides graphical features that aid supervisors in managing systems, such as visualizing weighted quality models, plotting data collected from managed systems, performing simulations, and management of adaptation rules. To accommodate the dashboard's functionalities, TMA's database schema had a few changes, and some of its components, such as the Application Programming Interface (API), the Analyze, and the Planning, also had to be changed.

Validation was performed on the dashboard's functionality, with black box testing techniques, and quality attributes. These attributes are usability, performance, and maintainability and, except for the latter which was based on following practices and conventions, tests were planned and executed. The validation results showed that within the tests performed the product is functionally valid and the performance requirements are satisfied. Concerning usability, it can be said the product is usable, although it presents a problem related to the clarity in the access to the feature that allows simulating metrics.

For future work, the final requirements assigned with the "Should" priority will be implemented to complement the support provided by the dashboard. Also, improvements in the usability of the product will be made, starting from the problem related to the simulation of metrics identified in the validation tests.

References

[1] Peter M. Mell and Timothy Grance. Sp 800-145. the nist definition of cloud computing. Technical report, Gaithersburg, MD, USA, 2011.

[2] Tharam Dillon, Chen Wu, and Elizabeth Chang. Cloud computing: Issues and challenges. In *2010 24th IEEE International Conference on Advanced Information Networking and Applications*, pages 27–33, 2010.

[3] What is virtualization? `https://opensource.com/resources/virtualization`. Accessed: 2021-12-01.

[4] Imran Ashraf. An overview of service models of cloud computing. *International journal of multidisciplinary and current research*, 2, 2014.

[5] Florian Auer, Valentina Lenarduzzi, Michael Felderer, and Davide Taibi. From monolithic systems to microservices: An assessment framework. *Information and Software Technology*, 137:106600, 2021.

[6] Fred Douglis and Jason Nieh. Microservices and containers. *IEEE Internet Comput.*, 23:5–6, 2019.

[7] Why should you use microservices and containers?, 2018. `https://developer.ibm.com/articles/why-should-we-use-microservices-and-containers/`. Accessed: 2021-12-15.

[8] What are containers? `https://www.netapp.com/devops-solutions/what-are-containers/`. Accessed: 2021-12-15.

[9] Container orchestration. `https://www.vmware.com/topics/glossary/content/container-orchestration`. Accessed: 2021-12-15.

[10] Production-grade container orchestration. `https://kubernetes.io/`. Accessed: 2022-01-21.

[11] Swarm mode overview. `https://docs.docker.com/engine/swarm/`. Accessed: 2022-01-21.

[12] Amazon elastic kubernetes service (eks). `https://aws.amazon.com/eks/`. Accessed: 2022-01-21.

[13] Azure kubernetes service (aks). `https://azure.microsoft.com/en-us/services/kubernetes-service/#overview`. Accessed: 2022-01-21.

[14] Top 12 advantages of effective it monitoring software, 2018. `https://www.cloudradar.io/blog/top-12-advantages-of-effective-it-monitoring`. Accessed: 2021-12-15.

[15] Nadeem Abbas, Jesper Andersson, and Danny Weyns. Asple: A methodology to develop self-adaptive software systems with systematic reuse. *Journal of Systems and Software*, 167:110626, 2020.

[16] Overview. `https://prometheus.io/docs/introduction/overview/`. Accessed: 2022-01-19.

[17] `https://grafana.com/`. Accessed: 2022-01-21.

[18] José Pereira, Rui Silva, Nuno Antunes, Jorge Silva, Breno Bernard França, Regina Moraes, and Marco Vieira. A platform to enable self-adaptive cloud applications using trustworthiness properties. pages 71–77, 06 2020.

[19] Spa (single-page application). `https://developer.mozilla.org/en-US/docs/Glossary/SPA`. Accessed: 2022-06-23.

[20] What is cloud computing? everything you need to know about the cloud explained. `https://www.zdnet.com/article/what-is-cloud-computing-everything-you-need-to-know-about-the-cloud/`. Accessed: 2021-12-27.

[21] 30 cloud monitoring tools: The definitive guide for 2021. `https://phoenixnap.com/blog/cloud-monitoring-tools`. Accessed: 2021-12-29.

[22] List of 13 best open source & free monitoring tools. `https://devopscube.com/best-opensource-monitoring-tools/`. Accessed: 2022-01-19.

[23] What is prometheus and why is it so popular? `https://www.cloudsavvyit.com/15124/what-is-prometheus-and-why-is-it-so-popular/`. Accessed: 2022-01-19.

[24] Overview. `https://graphiteapp.org/`. Accessed: 2022-01-19.

[25] Top 16 open source cloud monitoring tools in 2022. `https://roboticsbiz.com/top-16-open-source-cloud-monitoring-tools-in-2021/`. Accessed: 2022-01-19.

[26] Graphite. `https://www.aosabook.org/en/graphite.html`. Accessed: 2022-01-19.

[27] Overview. `https://graphite.readthedocs.io/en/latest/overview.html`. Accessed: 2022-01-19.

[28] Graphite vs. grafana: Build the best monitoring architecture for your application. `https://www.overops.com/blog/graphite-vs-grafana-build-the-best-monitoring-architecture-for-your-application/`. Accessed: 2022-01-19.

[29] What is zabbix? `https://www.educba.com/what-is-zabbix/`. Accessed: 2022-01-19.

[30] What is zabbix. https://www.zabbix.com/documentation/current/en/manual/introduction/about. Accessed: 2022-01-19.

[31] Zabbix overview. https://www.zabbix.com/documentation/current/en/manual/introduction/overview. Accessed: 2022-01-19.

[32] Zabbix architectures. https://subscription.packtpub.com/book/networking-and-servers/9781785289262/1/ch01lvl1sec09/zabbix-architectures. Accessed: 2022-01-19.

[33] Nagios tutorial: What is nagios tool? architecture & installation. https://www.guru99.com/nagios-tutorial.html. Accessed: 2022-01-20.

[34] Nagios - architecture. https://www.tutorialspoint.com/nagios/nagios_architecture.htm. Accessed: 2022-01-20.

[35] What is cadvisor? how does it work? explained.... https://www.scaleyourapp.com/what-is-cadvisor-how-does-it-work-explained/. Accessed: 2022-01-20.

[36] Containers metrics with prometheus and grafana. https://pramodshehan.medium.com/containers-metrics-in-prometheus-and-grafana-389555499eb8. Accessed: 2022-01-20.

[37] What are microservices? https://microservices.io/. Accessed: 2021-12-28.

[38] Microservices and containers. https://avinetworks.com/what-are-microservices-and-containers/.Accessed: 2021-12-15.

[39] Microservices vs monolith: which architecture is the best choice for your business? https://www.n-ix.com/microservices-vs-monolith-which-architecture-best-choice-your-business/. Accessed: 2021-12-28.

[40] Francisco Ponce Mella, Gastón Márquez, and Hernán Astudillo. Migrating from monolithic architecture to microservices: A rapid review. 09 2019.

[41] Microservices. https://martinfowler.com/articles/microservices.html. Accessed: 2021-12-28.

[42] Monolithfirst. https://martinfowler.com/bliki/MonolithFirst.html. Accessed: 2021-12-28.

[43] What are containers and why do you need them? https://www.cio.com/article/247005/what-are-containers-and-why-do-you-need-them.html. Accessed: 2022-01-18.

[44] Docker overview. https://docs.docker.com/get-started/overview/. Accessed: 2022-01-18.

[45] Docker architecture. https://www.aquasec.com/cloud-native-academy/docker-container/docker-architecture/. Accessed: 2022-01-18.

[46] What is kubernetes? https://www.redhat.com/en/topics/containers/what-is-kubernetes. Accessed: 2022-01-18.

[47] Pods. https://kubernetes.io/docs/concepts/workloads/pods/. Accessed: 2022-01-18.

[48] Kubernetes components. https://kubernetes.io/docs/concepts/overview/components/. Accessed: 2022-01-18.

[49] 10 best web development frameworks. https://hackr.io/blog/web-development-frameworks. Accessed: 2022-01-05.

[50] What is angular?: Architecture, features, and advantages. https://www.simplilearn.com/tutorials/angular-tutorial/what-is-angular. Accessed: 2022-01-05.

[51] John Kouraklis. *MVVM as Design Pattern*. 10 2016.

[52] Angular architecture overview. https://medium.com/@bhavikagarg8/angular-architecture-overview-1e7cc7483a0. Accessed: 2022-01-05.

[53] Is react a library or a framework? here's why it matters. https://www.freecodecamp.org/news/is-react-a-library-or-a-framework/. Accessed: 2022-01-05.

[54] Best front end frameworks for web development of 2021: The complete guide. https://medium.com/geekculture/best-front-end-frameworks-for-web-development-of-2021-the-complete-guide-ec30098fd1d0. Accessed: 2022-01-05.

[55] React: Create maintainable, high-performance ui components. https://developer.ibm.com/tutorials/wa-react-intro/. Accessed: 2022-01-06.

[56] What is jsx? https://www.reactenlightenment.com/react-jsx/5.1.html. Accessed: 2022-01-06.

[57] Introducing the new jsx transform. https://reactjs.org/blog/2020/09/22/introducing-the-new-jsx-transform.html. Accessed: 2022-01-06.

[58] Getting started. https://012.vuejs.org/guide/index.html Accessed: 2022-01-06.

[59] Vue.js is good, but is it better than angular or react? https://www.valuecoders.com/blog/technology-and-apps/vue-js-comparison-angular-react/ Accessed: 2022-01-06.

[60] What is vue.js? the pros and cons of vue.js in 2022. https://trio.dev/blog/why-use-vue-js. Accessed: 2022-01-06.

[61] What is vue.js? the pros and cons of vue.js framework. https://www.spaceo.ca/what-is-vue-js-and-its-pros-and-cons/. Accessed: 2022-01-21.

[62] What are the best frontend frameworks to use in 2021? https://www.ideamotive.co/blog/best-frontend-frameworks. Accessed: 2022-01-05.

[63] All about svelte, the much-loved, state-driven web framework. `https://thenewstack.io/all-about-svelte-the-much-loved-state-driven-web-framework/`. Accessed: 2022-01-07.

[64] Ember vs svelte: Comparing performance, architecture and more. `https://www.simform.com/blog/ember-vs-svelte/`. Accessed: 2022-01-12.

[65] Top 5 front-end javascript framework in 2022. `https://staticmania.com/blog/top-5-front-end-java-script-framework-in-2022`. Accessed: 2022-01-12.

[66] jquery. `https://www.javatpoint.com/what-is-jquery`. Accessed: 2022-01-12.

[67] Why outdated jquery is still the dominant javascript library. `https://thenewstack.io/why-outdated-jquery-is-still-the-dominant-javascript-library/`. Accessed: 2022-01-13.

[68] 11 benefits of jquery that every web designers should know of. `https://tekslate.com/11-benefits-jquery-every-web-designers-know`. Accessed: 2022-01-13.

[69] What is jquery: An intro for beginners. `https://www.coursereport.com/blog/what-is-jquery`. Accessed: 2022-01-13.

[70] Emberjs - overview. `https://www.tutorialspoint.com/emberjs/emberjs_overview.htm`. Accessed: 2022-01-13.

[71] 5 essential ember concepts. `https://emberigniter.com/5-essential-ember-concepts/`. Accessed: 2022-01-13.

[72] Introduction. `https://guides.emberjs.com/release/routing/`. Accessed: 2022-01-13.

[73] Rotan Sharma. Importance of ember.js and its advantages explained! `https://www.whatech.com/development/blog/590637-importance-of-ember-js-and-its-advantages-explained`. Accessed: 2022-01-13.

[74] Benefits of using emberjs for frontend web development. `https://coresumo.com/benefits-of-using-emberjs-for-frontend-web-development/`. Accessed: 2022-01-14.

[75] List of 10 best front end frameworks to use for web development. `https://www.monocubed.com/best-front-end-frameworks/`. Accessed: 2022-01-13.

[76] Angular, ember and vue: Is choosing a framework simply a matter of taste? `https://jaxenter.com/angular-ember-or-vue-choice-135987.html`. Accessed: 2022-01-13.

[77] Semantic ui guide. `https://www.freecodecamp.org/news/semantic-ui-guide/`. Accessed: 2022-01-17.

[78] Semantic ui. https://semantic-ui.com/. Accessed: 2022-01-17.

[79] Best css frameworks to look forward in 2021. https://www.lambdatest.com/blog/best-css-frameworks-2021/. Accessed: 2022-01-17.

[80] Yuqiong Sun, Susanta Nanda, and Trent Jaeger. Security-as-a-service for microservices-based cloud applications. In *2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 50–57, 2015.

[81] Elisabetta Di Nitto, Carlo Ghezzi, Andreas Metzger, Mike Papazoglou, and Klaus Pohl. A journey to highly dynamic, self-adaptive service-based applications. *Autom. Softw. Eng.*, 15:313–341, 12 2008.

[82] Vivek Nallur and Rami Bahsoon. A decentralized self-adaptation mechanism for service-based applications in the cloud. *IEEE Transactions on Software Engineering*, 39(5):591–612, 2013.

[83] David Sinreich. An architectural blueprint for autonomic computing. 2006.

[84] Christian Krupitzer, Felix Maximilian Roth, Sebastian VanSyckel, Gregor Schiele, and Christian Becker. A survey on engineering approaches for self-adaptive systems. *Pervasive Mob. Comput.*, 17:184–206, 2015.

[85] José Pereira, Rui Silva, Naghmeh Ivaki, Nuno Antunes, and Breno de França. D3.5 – monitoring instruments and platform implementation. https://www.atmosphere-eubrazil.eu/sites/default/files/ATMOSPHERE_D3_5_v2.pdf Accessed: 2021-12-28.

[86] Cornel Barna, Hamoun Ghanbari, Marin Litoiu, and Mark Shtern. Hogna: A platform for self-adaptive applications in cloud environments. In *2015 IEEE/ACM 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 83–87, 2015.

[87] Davi Monteiro Barbosa, Rômulo Gadelha De Moura Lima, Paulo Henrique Mendes Maia, and Evilásio Costa. Lotus@runtime: A tool for runtime monitoring and verification of self-adaptive systems. In *2017 IEEE/ACM 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 24–30, 2017.

[88] Nikolas Roman Herbst, Samuel Kounev, and Ralf Reussner. Elasticity in cloud computing: What it is, and what it is not. In *10th International Conference on Autonomic Computing (ICAC 13)*, pages 23–27, San Jose, CA, June 2013. USENIX Association.

[89] Google. Kubernetes Horizontal Pod Autoscaler, 2014. Accessed: 2022-02-20.

[90] Jóakim von Kistowski, Simon Eismann, Norbert Schmitt, André Bauer, Johannes Grohmann, and Samuel Kounev. TeaStore: A Micro-Service Reference Application for Benchmarking, Modeling and Resource Management

Research. In *Proceedings of the 26th IEEE International Symposium on the Modelling, Analysis, and Simulation of Computer and Telecommunication Systems*, MASCOTS '18, September 2018.

[91] Simon Eismann, Cor-Paul Bezemer, Weiyi Shang, Dušan Okanović, and André van Hoorn. Microservices: A performance tester's dream or nightmare? In *Proceedings of the ACM/SPEC International Conference on Performance Engineering*, ICPE '20, page 138–149, 2020.

[92] Sriyash Caculo, Kanishka Lahiri, and Subramaniam Kalambur. Characterizing the scale-up performance of microservices using teastore. In *2020 IEEE International Symposium on Workload Characterization (IISWC)*, pages 48–59, 2020.

[93] Xi Zhang, Alma Riska, and Erik Riedel. Characterization of the e-commerce storage subsystem workload. In *2008 Fifth International Conference on Quantitative Evaluation of Systems*, pages 297–306. IEEE, 2008.

[94] What is a use case? - definition & examples. `https://study.com/academy/lesson/what-is-a-use-case-definition-examples.html`. Accessed: 2022-01-15.

[95] Use cases. `https://www.usability.gov/how-to-and-tools/methods/use-cases.html`. Accessed: 2022-01-15.

[96] Sanford Friedenthal, Alan Moore, and Rick Steiner. Chapter 17 - residential security system example using the object-oriented systems engineering method. In Sanford Friedenthal, Alan Moore, and Rick Steiner, editors, *A Practical Guide to SysML (Third Edition)*, The MK/OMG Press, pages 417–504. Morgan Kaufmann, Boston, third edition edition, 2015.

[97] Nonfunctional requirements. `https://www.scaledagileframework.com/nonfunctional-requirements/`. Accessed: 2022-01-17.

[98] How fast should my website load? `https://www.dotcom-tools.com/web-performance/blog/how-fast-should-my-website-load/`. Accessed: 2022-02-10.

[99] The c4 model for visualising software architecture. `https://c4model.com/`. Accessed: 2022-01-24.

[100] The c4 model for software architecture. `https://www.infoq.com/articles/C4-architecture-model/`. Accessed: 2022-01-24.

[101] Mohd Ehmer and Farmeena Khan. A comparative study of white box, black box and grey box testing techniques. *International Journal of Advanced Computer Science and Applications*, 3, 06 2012.

[102] Asma Bhat and S. M. K. Quadri. Equivalence class partitioning and boundary value analysis - a review. In *2015 2nd International Conference on Computing for Sustainable Global Development (INDIACom)*, pages 1557–1562, 2015.

[103] Semantic differential scale: Definition, examples. https://www.statisticshowto.com/semantic-differential-scale/. Accessed: 2022-06-27.

[104] Manick Bhan. A technical guide to google's pagespeed insights reports, 2018. https://www.statisticshowto.com/semantic-differential-scale/. Accessed: 2022-06-30.

[105] Product development: The waterfall methodology (model) in software development. https://learn.marsdd.com/article/product-development-the-waterfall-methodology-model-in-software-development/. Accessed: 2022-01-21.

[106] Sdlc - waterfall model. https://www.tutorialspoint.com/sdlc/sdlc_waterfall_model.htm. Accessed: 2022-01-21.

[107] Moscow method. https://www.projectsmart.co.uk/tools/moscow-method.php. Accessed: 2022-01-17.

[108] Moscow prioritization. https://www.productplan.com/glossary/moscow-prioritization/. Accessed: 2022-01-17.

[109] Christopher Alberts and Audrey Dorofee. Risk management framework. Technical Report CMU/SEI-2010-TR-017, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 2010.

Appendices

Appendix A

Use cases details

In this document, the multiple use cases identified in the diagrams are described in detail. Here, the use cases are organized by categories where they most belong. Those categories are: Quality Models, Metrics, Resources, Probes, Actuators, Actions, Adaptation Rules, Plans and Logs.

Following there is a description of each category that resumes in a simple way what it represents:

1. **Metrics:** Metrics are what compose the Quality Models. They represent properties of a system and thereby they are what is monitored.

2. **Quality Models:** Quality Models define hierarchical trees of metrics that may have available multiple configuration profiles which define weights for each metric. Selecting a Quality Model, and consequently a configuration profile for it, originates a weighted tree that can then be used to calculate scores to reflect a system's state. So, this state is represented through a number, score, that will reflect the health of a system according to a defined Quality Model.

3. **Descriptions:** Descriptions are used to identify and describe data collected by Probes. Also, Metrics which are leaf attributes on a Quality Model tree use these descriptions to describe the data they are related to.

4. **Resources:** Resources are the monitored subjects. Metrics are gathered from these objects. A system that is being monitored may have several resources or even be composed by a single resource.

5. **Probes:** Probes are the components in charge of gathering raw information from systems states. Basically, they collect raw data from monitored resources which is then used to compute the metrics values.

6. **Actuators:** Actuators are responsible for executing adaptations on a system. Thereby they are what assures adaptation plans are executed.

7. **Actions:** Actions define what is going to be changed by actuators and in which part of the system (resource). Actions have configurations which are

the parameters to be changed. There might be different actions that change the same configurations but in different values.

8. **Adaptation Rules:** Adaptation Rules create plans to be executed when a condition is triggered. This condition is a comparison between gathered metrics' data and some threshold value. TMA's Planning is the component responsible for verifying these rules.
9. **Plans:** Plans are created by adaptation rules and they represent an ordered execution of actions. These Plans are orchestrated by TMA's Execute component.
10. **Logs:** Logs are important events on TMA's platform that need to be saved in order to provide an alternative view on its management.
11. **Dashboard:** This section presents a dashboard's inherent functionalities that become important in the context of TMA.

A.1 Metrics

A.1.1 Use Case - View Metric information

Table A.1: Use Case 1.1 - View Metric information

Name:	View Metric information
Primary Actor:	Administrator
Priority:	Must
Stakeholders and Interests:	Administrator wants to view a Metric's metadata information and corresponding tree of metrics
Preconditions:	Administrator navigates on the dashboard to the Metrics' section and selects the option to view information
Main Success scenario:	1. The System presents a list of metrics 2. Administrator selects a Metric 3. System displays the id, name, block level of the selected Metric and triggers Use Case - Preview metrics tree 4. **Extension Points:** Use Case - Update Metric / Use Case - Delete Metric

Extensions:	1a. Database is inaccessible: • 1a.1 The System presents a message, saying that the list of metrics could not be retrieved because database is inaccessible • 1a.2 Administrator is redirected to the homepage 3a. Metric is a Leaf Attribute: • 3a.1 System displays name, block level, aggregation operator, description associated, number of samples, normalization method, normalization kind, minimum threshold, maximum threshold and an indication that the metric is a leaf attribute • 3a.2 Use case continues at step 4
Minimal guarantee:	Metric's information can not be seen
Success guarantee:	Detailed information on a Metric can be seen, with extra information in case it is a leaf attribute

A.1.2 Use Case - Create Metric

Table A.2: Use Case 1.2 - Create Metric

Name:	Create Metric
Primary Actor:	Administrator
Priority:	Must
Stakeholders and Interests:	Administrator wants to add a metric to TMA
Preconditions:	Administrator navigates on the dashboard to the Metric's section and selects the option to create

Main Success scenario:	1. System presents a form with name, block level and an option of setting the metric to leaf attribute to be completed 2. System triggers Use Case - Associate child metrics 3. Administrator fills the form 4. System verifies that the option for creating a leaf attribute is not set, the name is unique and not empty, and that block level is not empty 5. System creates the metric 6. System presents a message saying that the metric was created 7. System redirects Administrator to homepage
Extensions:	2a. Administrator selects option for setting the metric as a leaf attribute: • 2a.1 **Extension Point:** Use Case - Create Leaf Attribute Metric • 2a.2 Use case continues at step 5 4a. Block level/name is empty or name is not unique: • 4a.1 The system presents a message, saying that the name field can not be empty or that it already exists, or that block level can not be empty, depending on what generated this extension • 4a.2 The use continues at step 3 4b. The database is inaccessible: • 4b.1 The System presents a message, saying that the name introduced could not be verified to be unique because database is inaccessible • 4b.2 Use case continues at step 7
Minimal guarantee:	TMA's database remains the same when it comes to metrics
Success guarantee:	A new Metric is added to the TMA's database

A.1.3 Use Case - Create Leaf Attribute Metric

Table A.3: Use Case 1.3 - Create Leaf Attribute Metric

Name:	Create Leaf Attribute Metric
Primary Actor:	Administrator
Priority:	Must
Stakeholders and Interests:	Administrator wants to add a leaf attribute metric to TMA
Preconditions:	System triggered Use Case - Create Metric
Main Success scenario:	1. System presents another form fields to be completed: aggregation operator, number of samples, normalization method, normalization kind, minimum threshold, maximum threshold fields to be completed 2. Administrator fills the form 3. System triggers Use Case - Associate Description to Leaf Attribute Metric 4. System verifies that the fields of the form are not empty and match the type and restrictions
Extensions:	4a. Fields are empty or do not match their types and restrictions: • 4a.1 The system presents a message, informing the type of input required, depending on what generated this extension • 4a.2 The use continues at step 2
Minimal guarantee:	System will hold the inserted field inputs
Success guarantee:	Form fields of Leaf Attribute Metric are filled and the association to a description made

A.1.4 Use Case - Update Metric

Table A.4: Use Case 1.4 - Update Metric

Name:	Update Metric
Primary Actor:	Administrator
Priority:	Won't

Stakeholders and Interests:	Administrator wants to update the information of a metric
Preconditions:	System triggered Use Case - View Metric Information
Main Success scenario:	1. Administrator selects the option to edit 2. The System presents editable information on the name field 3. **Extension point:** Use Case - Associate child metrics 4. Administrator updates metric's name 5. The system verifies that the name field is unique and not empty 6. The system updates the metric 7. The system presents a message, saying the metric was successfully updated

Extensions:	2a. The metric being updated is a LeafAttribute: • 2a.1 The system presents, besides the name, editable aggregation operator, number of samples, normalization method, normalization kind, minimum threshold and maximum threshold fields • 2a.2 Administrator updates name, metric's aggregation operator, number of samples, normalization method, normalization kind, minimum threshold and maximum threshold • 2a.3 The System verifies that the updated fields are not empty and match the type and length restrictions of the database • 2a.4 Use case continues at step 5 5a. Name field is empty or already exists: • 5a.1 The system presents a message, saying that the name field can not be empty or that it already exists, depending on what generated this extension • 5a.2 The use continues at step 2 6a. Database is inaccessible: • 6a.1 The System presents a message, saying that the metric could not be updated because database is inaccessible • 6a.2 System redirects Administrator to the homepage
Minimal guarantee:	Selected metric's information remains the same as the beginning of the use case
Success guarantee:	The selected metric's information is updated

A.1.5 Use Case - Delete Metric

Table A.5: Use Case 1.5 - Delete Metric

Name:	Delete metric
Primary Actor:	Administrator
Priority:	Won't

Stakeholders and Interests:	Administrator wants to delete a metric that has never been used and that is not a parent of another metric
Preconditions:	System triggered Use Case - View Metric information Selected Metric has no MetricData associated in the database Selected Metric does not have a parent Metric Selected Metric is not associated to a Quality Model
Main Success scenario:	1. Administrator selects the option to delete 2. System deletes selected metric 3. System presents a message saying that the metric was deleted 4. System redirects Administrator to the homepage
Extensions:	2a. TMA's database is inaccessible: • 2a.1 The System presents a message, saying that the metric could not be deleted because database is inaccessible • 2a.2 Use case continues at step 4
Minimal guarantee:	None metric is deleted from TMA's database
Success guarantee:	The deletion of a selected Metric, that has no associated data, is not a parent metric and is not associated to a Quality model, is accomplished

A.1.6 Use Case - Associate Description to Leaf Attribute Metric

Table A.6: Use Case 1.6 - Associate Description to Leaf Attribute Metric

Name:	Associate Description to Leaf Attribute Metric
Primary Actor:	Administrator
Priority:	Must
Stakeholders and Interests:	Administrator wants to associate the description that the metric being created, which is a leaf attribute, is about
Preconditions:	

Main Success scenario:	1. System presents a list of descriptions 2. Administrator selects a description 3. System presents a preview on the name, data type and unit information of the selected description 4. Administrator confirms the selection 5. System associates selected description to the metric being created
Extensions:	1a. Database is inaccessible: • 1a.1 The System presents a message, saying that the list of metrics could not be retrieved because database is inaccessible • 1a.2 System redirects Administrator to the homepage 3a. Database is inaccessible: • 3a.1 The System presents a message, saying that the details of the selected description could not be retrieved because database is inaccessible • 3a.2 System redirects Administrator to the homepage 4a. Administrator wants to change its description selection: • 4a.1 Use case continues at step 2
Minimal guarantee:	
Success guarantee:	Selected description is associated to the metric being created

A.1.7 Use Case - Associate Child Metrics

Table A.7: Use Case 1.7 - Associate Child Metrics

Name:	Associate Child Metrics
Primary Actor:	Administrator
Priority:	Must

Stakeholders and Interests:	Administrator wants to associate child metrics to a parent one
Preconditions:	• System triggered Use Case - Create metric or Use Case - Update Metric • Metric is not associated to a Quality Model that has been used
Main Success scenario:	1. System presents a list of metrics 2. Iteratively, Administrator selects metrics from the list to be associated as childs of the current metric and triggers Use Case - Preview metrics tree 3. System associates selected metrics as childs of the current
Extensions:	1a. Database is inaccessible: • 1a.1 The System presents a message, saying that the metrics information could not be retrieved because database is inaccessible • 1a.2 System redirects Administrator to the homepage 1b. When updating a metric: • 1b.1 System presents a list of metrics without the ones that are already associated and triggers Use Case - Preview metrics • 1b.2 Use case continues at step 2 2a. Administrator wants to remove an associated child metric: • 2a.1 Administrator removes metric from list of child metrics to be associated and triggers Use Case - Preview metrics • 2a.2 Use case continues at step 2
Minimal guarantee:	Child metric associations can not be performed
Success guarantee:	Child metrics are associated to a metric

A.1.8 Use Case - Preview metrics tree

Table A.8: Use Case 1.8 - Preview metrics tree

Name:	Preview metrics tree
Primary Actor:	Administrator
Priority:	Must
Stakeholders and Interests:	Administrator wants to graphically visualize the tree composition of a root metric
Preconditions:	• There is at least one metric • A root node/metric is received
Main Success scenario:	1. System loads information of the tree structure for the root node given 2. System presents graphically, a tree structure of the metrics tree, identifying which metric goes in which node of the tree
Extensions:	1a. Database is inaccessible: • 1a.1 The System presents a message, saying that the tree structure for the root node given could not be retrieved because database is inaccessible • 1a.2 System redirects Administrator to the homepage
Minimal guarantee:	Preview of the metrics tree can not be seen
Success guarantee:	A tree of metrics can be visualized

A.2 Quality Models

A.2.1 Use Case - View Quality Model Information

Table A.9: Use Case 2.1 - View Quality Model Information

Name:	View Quality Model Information
Primary Actor:	Administrator

Priority:	Must
Stakeholders and Interests:	Administrator wants to view a Quality Model's metadata and metrics tree
Preconditions:	Administrator navigates on the dashboard to the Quality Model's section and selects the option to see detailed information.
Main Success scenario:	1. The system presents a list of Quality Models 2. Administrator selects a Quality Model 3. The System presents the id, name, description, and list of associated configuration profiles of the selected Quality Model and triggers Use Case - Preview metrics tree 4. **Extension Points:** Use Case - Update Quality Model / Use Case - Delete Quality Model / Use Case - View Configuration Profile Information / Use Case - Create Configuration Profile
Extensions:	1a. / 3a. Database is inaccessible: • 1a.1 / 3a.1 The System presents a message, saying that the Quality Model's information could not be retrieved because database is inaccessible • 1a.2 / 3a.2 The System redirects Administrator to homepage
Minimal guarantee:	Quality Model's information can not be seen
Success guarantee:	The metadata and tree of metrics of the selected Quality Model is shown

A.2.2 Use Case - Create Quality Model

Table A.10: Use Case 2.2 - Create Quality Model

Name:	Create Quality Model
Primary Actor:	Administrator
Priority:	Must
Stakeholders and Interests:	Administrator wants to create a Quality Model
Preconditions:	Administrator navigates on the dashboard to the Quality Model's section and selects the option for creating.

Main Success scenario:	1. System presents a form with name and description fields to be completed and triggers Use Case - Associate Metric to Quality Model 2. Administrator fills the form 3. The System verifies that the name field is unique and not empty, and that a metric has been associated to the quality model being created 4. The system persists the Quality Model 5. The system presents a message saying that the Quality Model was created 6. The system redirects the Administrator to the homepage
Extensions:	3a. Name field is empty or already exists: • 3a.1 The system presents a message, saying that the name field can not be empty or that it already exists, depending on what generated this extension • 3a.2 The use continues at step 2 3b. Administrator did not associate a metric to the quality model being created: • 3b.1 The system presents a message, saying that a metric must be associated to the quality model • 3b.2 The use continues at step 4 5a. Database is inaccessible: • 5a.1 The System presents a message, saying that the quality model could not be created because database is inaccessible • 5a.2 The use case continues at step 8
Minimal guarantee:	No Quality Model is created
Success guarantee:	The Quality Model is created and persisted at TMA's database

A.2.3 Use Case - Update Quality Model

Table A.11: Use Case 2.3 - Update Quality Model

Name:	Update Quality Model
Primary Actor:	Administrator
Priority:	Won't
Stakeholders and Interests:	Administrator wants to update metadata of a Quality Model
Preconditions:	System triggered Use Case - View Quality Model
Main Success scenario:	1. The Administrator selects the option to edit the Quality Model 2. The System presents editable information on the name and description fields of the Quality Model 3. Administrator updates Quality Model's name and description 4. The system verifies that the name field is unique and not empty 5. The system updates the Quality Model 6. The system presents a message, saying the quality model was updated
Extensions:	4a. Name field is empty or already exists: • 4a.1 The system presents a message, saying that the name field can not be empty or that it already exists, depending on what generated this extension • 4a.2 The use continues at step 3 5a. Database is inaccessible: • 5a.1 The System presents a message, saying that the quality model could not be updated because database is inaccessible • 5a.2 System redirects Administrator to the homepage
Minimal guarantee:	There is no change on the selected Quality Model
Success guarantee:	The selected Quality Model is updated

A.2.4 Use Case - Delete Quality Model

Table A.12: Use Case 2.4 - Delete Quality Model

Name:	Delete Quality Model
Primary Actor:	Administrator
Priority:	Won't
Stakeholders and Interests:	Administrator wants to delete a Quality Model that has never been used, or that is not being used.
Preconditions:	• System triggered Use Case - View Quality Model Information • Selected Quality Model has not been used once or is not being used
Main Success scenario:	1. Administrator chooses the option to delete 2. System deletes the selected Quality Model 3. System presents a message, saying that the Quality Model was deleted 4. System redirects Administrator to the homepage
Extensions:	2a. Database is inaccessible: • 2a.1 The System presents a message, saying that the Quality Model could not be deleted because database is inaccessible • 2a.2 The use case continues at step 4
Minimal guarantee:	None Quality Model is deleted from TMA's database
Success guarantee:	The deletion of a selected Quality Model, which has not been used once, is accomplished

A.2.5 Use Case - View Configuration Profile Information

Table A.13: Use Case 2.5 - View Configuration Profile Information

Name:	View Configuration Profile Information
Primary Actor:	Administrator
Priority:	Must

Stakeholders and Interests:	Administrator wants to view a Quality Model's configuration profile information of his
Preconditions:	Use Case - View Quality Model Information
Main Success scenario:	1. Administrator selects a configuration profile 2. System triggers Use Case - Preview metrics tree and, to its outcome, adds the weights of the selected configuration profile 3. **Extension Points:** Use Case - Update Configuration Profile / Use Case - Delete Configuration Profile
Extensions:	1a. / 3a. Database is inaccessible: • 1a.1 / 3a.1 The System presents a error message, saying that the database is inaccessible • 1a.2 / 3a.2 The System redirects Administrator to homepage
Minimal guarantee:	Configuration profile's weights for the Quality Model's metrics tree is not shown
Success guarantee:	Configuration profile's weights for the Quality Model's metrics tree can be seen

A.2.6 Use Case - Create Configuration Profile

Table A.14: Use Case 2.6 - Create Configuration Profile

Name:	Create Configuration Profile
Primary Actor:	Administrator
Priority:	Must
Stakeholders and Interests:	Administrator wants to create a configuration profile for a Quality Model
Preconditions:	Use Case - View Quality Model Information

Main Success scenario:	1. Administrator selects the option for creating a configuration profile for the selected Quality Model 2. The system presents a name field to be filled 3. System triggers Use case - Preview metrics tree and presents a form to fill in the weights for the tree 4. Administrator fills in the name and the weights fields 5. System verifies that a name was written, it is not used by another configuration profile for the same quality model and that a node's child weights is equal to 1 6. The system persists the association of the configuration profile to the Quality Model 7. The system presents a success message informing the configuration profile was created 8. The system triggers Use case - View Quality Model Information

Extensions:	5a. Administrator does not fill in the configuration profile name: • 5a.1 The system presents a message asking for the user to introduce a name for the configuration profile • 5a.2 The use case continues at step 4 5b. Administrator introduces a name in use by another configuration profile for the same quality model: • 5b.1 The system presents a message asking for the user to introduce another name, because the introduced one is already in use • 5b.2 The use case continues at step 4 5c. The Administrator incorrectly fills in the configuration profile weights: • 5c.1 The system presents information about the error nature • 5c.2 The use case continues at step 4 6a. Database is inaccessible: • 6a.1 The System presents a message, saying that the configuration profile could not be created because database is inaccessible • 6a.2 The use case continues at step 8
Minimal guarantee:	No configuration profile is created
Success guarantee:	The configuration profile is created and associated to a Quality Model at TMA's database

A.2.7 Use Case - Update Configuration Profile

Table A.15: Use Case 2.7 - Update Configuration Profile

Name:	Update Configuration Profile
Primary Actor:	Administrator
Priority:	Won't
Stakeholders and Interests:	Administrator wants to update a configuration profile's name and nodes weights
Preconditions:	Use Case - View Configuration Profile

Main Success scenario:	1. The Administrator selects the option to edit the configuration profile 2. System triggers Use Case - Preview metrics tree 3. The System presents editable information on the configuration profile's name and the weights assigned to each metrics tree node 4. Administrator updates configuration profile's name and weights 5. The system verifies that the name is not null and that each node's weight is not 0 and that a node's child weights is equal to 1 6. The system updates the configuration profile 7. The system presents a message, saying the update was successful 8. The system redirects the Administrator to the homepage
Extensions:	5a. Name field was submitted as null: • 5a.1 The system presents a message, saying that the name for the configuration profile must be filled • 5a.2 The use continues at step 4 5b. Administrator incorrectly fills in the configuration profile weights: • 5b.1 The system presents information about the error nature • 5b.2 The use continues at step 4 6a. Database is inaccessible: • 6a.1 The System presents a error message, saying that the database is inaccessible • 6a.2 The use case continues at step 9
Minimal guarantee:	There is no change on the selected configuration profile
Success guarantee:	The selected configuration profile has its name and nodes weights updated

A.2.8 Use Case - Delete Configuration Profile

Table A.16: Use Case 2.8 - Delete Configuration Profile

Name:	Delete Configuration Profile
Primary Actor:	Administrator
Priority:	Won't
Stakeholders and Interests:	Administrator wants to delete a configuration profile of a Quality Model that is not being used
Preconditions:	• Use Case - View Configuration Profile • Selected configuration profile is not being used
Main Success scenario:	1. Administrator chooses the option to delete 2. System deletes the selected configuration profile from TMA's database 3. System presents a message, saying that the configuration profile was successfully deleted 4. The system redirects the Administrator to the homepage
Extensions:	2a. Database is inaccessible: • 2a.1 The System presents a error message, saying that the database is inaccessible • 2a.2 The use case continues at step 4
Minimal guarantee:	Configuration profile is not deleted from TMA's database
Success guarantee:	The deletion of the selected configuration profile is accomplished

A.2.9 Use Case - Associate Metric to Quality Model

Table A.17: Use Case 2.9 - Associate Metric to Quality Model

Name:	Associate Metric to Quality Model
Primary Actor:	Administrator
Priority:	Must

Stakeholders and Interests:	Administrator wants to define the root node of the metrics tree that represents the Quality Model
Preconditions:	• There is at least one metric • System triggered Use Case - Create Quality Model
Main Success scenario:	1. System presents a list of metrics 2. Administrator selects a metric 3. System triggers Use Case - Preview metrics tree 4. System associates selected metric to the quality model being created
Extensions:	1a. Database is inaccessible: • 1a.1 The System presents a message, saying that the list of metrics could not be retrieved because database is inaccessible • 1a.2 System redirects Administrator to the homepage 3a. Administrator wants to change its metric selection: • 3a.1 Administrator selects another metric • 3a.2 Use case continues at step 3
Minimal guarantee:	System holds the value of the last selected metric
Success guarantee:	Selected metric is associated to the quality model being created

A.3 Descriptions

A.3.1 Use Case - View Description Information

Table A.18: Use Case 3.1 - View Description Information

Name:	View Description Information
Primary Actor:	Administrator
Priority:	Could

Stakeholders and Interests:	Administrator wants to view a Description's information
Preconditions:	Administrator navigates on the dashboard to the Description's section and selects the option to view information
Main Success scenario:	1. System presents a list of Descriptions 2. Administrator selects a Description 3. System displays id, datatype, name and unit of the selected Description 4. **Extension Points:** Use Case - Update Description/ Use Case - Delete Description
Extensions:	1a. Database is inaccessible: • 1a.1 System presents a message, saying that the descriptions could not be gathered because database is inaccessible • 1a.2 System redirects Administrator to the homepage
Minimal guarantee:	Description's information can not be seen
Success guarantee:	Detailed information on a Description can be seen

A.3.2 Use Case - Create Description

Table A.19: Use Case 3.2 - Create Description

Name:	Create Description
Primary Actor:	Administrator
Priority:	Could
Stakeholders and Interests:	Administrator wants to add a description to TMA
Preconditions:	Administrator navigates on the dashboard to the Description's section and selects the option to create

Main Success scenario:	1. System presents a form with datatype, name and unit fields to be completed 2. Administrator fills the form 3. System verifies that the form was well filled 4. System creates the description with given datatype, name and unit 5. System presents a message saying that the description was created 6. System redirects Administrator to homepage
Extensions:	3a. Some parameter is not correctly filled: • 3a.1 The system verifies that a parameter was badly defined • 3a.2 The system presents a message, saying that a parameter was badly defined, along with information on why • 3a.3 The use continues at step 2 4a. The database is inaccessible: • 4a.1 The System presents a message, saying that the description could not be created because database is inaccessible • 4a.2 Use case continues at step 6
Minimal guarantee:	TMA's database remains the same when it comes to descriptions
Success guarantee:	A new Description is added to the TMA's database

A.3.3 Use Case - Update Description

Table A.20: Use Case 3.3 - Update Description

Name:	Update Description
Primary Actor:	Administrator
Priority:	Won't
Stakeholders and Interests:	Administrator wants to update the information of a description

Preconditions:	System triggered Use Case - View Description Information
Main Success scenario:	1. Administrator selects the option to edit 2. The System presents editable information on the datatype, name and unit of the selected description 3. Administrator updates description's information 4. System verifies that all of description's information were well set 5. The System updates the description's information 6. System presents a message saying that the description was updated 7. System redirects Administrator to the homepage
Extensions:	4a. Some parameter is not correctly filled: • 4a.1 The system verifies that a parameter was badly defined • 4a.2 The system presents a message, saying that a parameter was badly defined, along with information on why • 4a.3 The use continues at step 3 5a. TMA's database is inaccessible: • 5a.1 The System presents a message, saying that the description could not be updated because database is inaccessible • 5a.2 Use case continues at step 7
Minimal guarantee:	Selected description's information remains the same as the beginning of the use case
Success guarantee:	The selected description's information is updated

A.3.4 Use Case - Delete Description

Table A.21: Use Case 3.4 - Delete Description

Name:	Delete Description
Primary Actor:	Administrator
Priority:	Won't
Stakeholders and Interests:	Administrator wants to delete a description
Preconditions:	System triggered Use Case - View Description Information
Main Success scenario:	1. Administrator selects the option to delete 2. System deletes selected description 3. System presents a message saying the description was deleted 4. System redirects Administrator to the homepage
Extensions:	2a. TMA's database is inaccessible: • 2a.1 The System presents a message, saying that the description could not be deleted because database is inaccessible • 2a.2 Use case continues at step 4
Minimal guarantee:	None description is deleted from TMA's database
Success guarantee:	The selected description is deleted from TMA's database

A.4 Resources

A.4.1 Use Case - View Resource information

Table A.22: Use Case 4.1 - View Resource information

Name:	View Resource information
Primary Actor:	Administrator
Priority:	Could
Stakeholders and Interests:	Administrator wants to view a Resource's information
Preconditions:	Administrator navigates on the dashboard to the Resource's section and selects the option to view information

Main Success scenario:	1. System presents a list of resources 2. Administrator selects a resource 3. System displays id, name, type and address of the selected Resource 4. **Extension Points:** Use Case - Update Resource / Use Case - Delete Resource
Extensions:	1a. Database is inaccessible: • 1a.1 System presents a message, saying that the resources could not be gathered because database is inaccessible • 1a.2 System redirects Administrator to the homepage
Minimal guarantee:	None kind of information can be seen
Success guarantee:	Detailed information on a Resource can be seen

A.4.2 Use Case - Create Resource

Table A.23: Use Case 4.2 - Create Resource

Name:	Create Resource
Primary Actor:	Administrator
Priority:	Could
Stakeholders and Interests:	Administrator wants to add a Resource to TMA
Preconditions:	Administrator navigates on the dashboard to the Resource's section and selects the option to create

Main Success scenario:	1. System presents a form with name, type and address fields to be filled in 2. Administrator fills the form 3. System triggers Use Case - Associate Configuration Profile to Resource 4. System verifies that the form was well filled and that the association to a configuration profile was done 5. System creates the resource with given name, type and address 6. System presents a message saying that the resource was created 7. System redirects Administrator to homepage
Extensions:	4a. Some form parameter is not correctly filled: • 4a.1 The system verifies that a parameter was badly defined • 4a.2 The system presents a message, saying that a parameter was badly defined, along with information on why • 4a.3 The use continues at step 2 4b. Configuration Profile's association is missing: • 4b.1 System presents a message, saying that a configuration profile association must be performed • 4b.2 The use continues at step 3 5a. The database is inaccessible: • 5a.1 The System presents a message, saying that the resource could not be created because database is inaccessible • 5a.2 Use case continues at step 7
Minimal guarantee:	TMA's database remains the same when it comes to resources
Success guarantee:	A new Resource is added to the TMA's database

A.4.3 Use Case - Update Resource

Table A.24: Use Case 4.3 - Update Resource

Name:	Update Resource
Primary Actor:	Administrator
Priority:	Won't
Stakeholders and Interests:	Administrator wants to update the information of a resource
Preconditions:	System triggered Use Case - View Resource Information
Main Success scenario:	1. Administrator selects the option to edit 2. The System presents editable information on the name, type and address fields of the selected resource 3. Administrator updates resource's information 4. **Extension Point:** Use Case - Associate Configuration Profile to Resource 5. System verifies that all of resource's information were well set 6. The System updates the resource's information 7. System presents a message saying that the resource was updated 8. System redirects Administrator to the homepage

Extensions:	4a. Configuration Profile's association is missing: • 4a.1 System verifies that configuration profile association is missing • 4a.2 System presents a message, saying that a configuration profile association must be set • 4a.3 Use case continues at step 4 5a. Some form parameter is not correctly filled: • 5a.1 The system presents a message, saying that with information on why a parameter is badly defined • 5a.2 Use case continues at step 3 6a. TMA's database is inaccessible: • 6a.1 The System presents a message, saying that the resource could not be updated because database is inaccessible • 6a.2 Use case continues at step 8
Minimal guarantee:	Selected resource's information remains the same as the beginning of the use case
Success guarantee:	The selected resource's information is updated

A.4.4 Use Case - Delete Resource

Table A.25: Use Case 4.4 - Delete Resource

Name:	Delete resource
Primary Actor:	Administrator
Priority:	Won't
Stakeholders and Interests:	Administrator wants to delete a resource
Preconditions:	System triggered Use Case - View Resource information

Main Success scenario:	1. Administrator selects the option to delete 2. System deletes selected resource 3. System presents a message saying that the resource was deleted 4. System redirects Administrator to the homepage
Extensions:	2a. TMA's database is inaccessible: • 2a.1 The System presents a message, saying that the resource could not be deleted because database is inaccessible • 2a.2 Use case continues at step 4
Minimal guarantee:	None resource is deleted from TMA's database
Success guarantee:	The selected resource is deleted from TMA's database

A.4.5 Use Case - Visualize Resource Metrics

Table A.26: Use Case 4.5 - Visualize Resource Metrics

Name:	Visualize Resource Metrics
Primary Actor:	Administrator
Priority:	Must
Stakeholders and Interests:	Administrator wants to visualize, graphically, probing values gathered from a managed resource
Preconditions:	Administrator navigates on the dashboard to the resources' section and selects the option to visualize resource metrics

Main Success scenario:	1. The system loads from TMA's database the list of resources associated to the Administrator 2. Administrator selects a resource 3. System triggers Use Case - Preview metrics tree with the root node of the metrics tree associated to the selected resource 4. System presents the list of metrics that compose the metrics tree 5. Administrator selects a metric from the list 6. System awaits an input for the live time window 7. Administrator introduces the time window for visualizing metric values being collected 8. The system presents, in real time, the selected metric's values being collected from the chosen resource in the specified time window 9. **Extension Points:** Use Case - Plot plans alongside metrics, Use Case - Export charts, Use Case - Simulate resource metrics, Use Case - Export chart config

Extensions:	2a. The Administrator wants to change the selected resource: • 2a.1 The Administrator changes the resource selection • 2a.2 The use case continues at step 3 5a. The Administrator wants to change metric: • 5a.1 The Administrator selects other metric • 5a.2 The use case continues at step 6 6a. The Administrator does not want to see live metrics being collected, but instead the values gathered in a time interval: • 6a.1 System provides a form to specify a start and end timestamps • 6a.2 Administrator provides the timestamps • 6a.3 The system presents the selected metric's values collected from the chosen resource in the specified time interval • 6a.4 Use case continues at step 9 7a. The Administrator wants to change introduced live time window: • 7a.1 The Administrator introduces new time window • 7a.2 The use case continues at step 8 8a. The system can not get the values due to some error: • 8a.1 The system alerts the Administrator that the values could not be gathered and suggests trying it later. • 8a.2 The system redirects Administrator to the homepage
Minimal guarantee:	Nothing is shown to the Administrator
Success guarantee:	The System presents to the Administrator live values from a resource's metric in a specified time window

A.4.6 Use Case - Simulate Resource Metrics

Table A.27: Use Case 4.6 - Simulate Resource Metrics

Name:	Simulate Resource Metrics
Primary Actor:	Administrator
Priority:	Must
Stakeholders and Interests:	Looking at collected metrics presented in a chart, Administrator may find util simulating metrics values in order to analyze what would be the behavior of the system with other weights applied to the metrics tree
Preconditions:	System triggered Use Case - Visualize resource metrics
Main Success scenario:	1. Administrator selects the option to simulate resource metrics 2. System saves the current time interval and metrics values of the current chart 3. System presents the configuration profile's nodes and its weights in an editable manner, and the list of metrics that match the nodes 4. Administrator alters some of the weights to perform a simulation 5. Administrator confirms the operation to simulate 6. System recalculates values of the metric within the saved time interval for the selected resource, and presents the results in a new chart 7. **Extension Point:** Use Case - Export charts
Extensions:	3a. TMA's database is inaccessible: • 3a.1 The System presents a message, saying that the weights of the configuration profile could not be gathered because database is inaccessible • 3a.2 Administrator is redirected to the homepage 4a. The Administrator wants to rechange weights that he already changed: • 4a.1 The Administrator rechanges the weights • 4a.2 The use case continues at step 5

Minimal guarantee:	Simulating a different weighted tree can not be done
Success guarantee:	The System presents to the Administrator the simulation's chart of applying a different weighted tree on the resource's metrics collected

A.4.7 Use Case - Plot Plans Alongside Metrics

Table A.28: Use Case 4.7 - Plot Plans Alongside Metrics

Name:	Plot Plans Alongside Metrics
Primary Actor:	Administrator
Priority:	Must
Stakeholders and Interests:	Administrator wants to visualize, besides metrics values, plans that took place
Preconditions:	System triggered Use Case - Visualize resource metrics
Main Success scenario:	1. Administrator selects the option to identify plans that took place in the time interval 2. The system loads from TMA's database the list of plans that were performed in the graphic time interval 3. The system presents in addition to the metric values, the plans that occurred in the time interval
Extensions:	2a. TMA's database is inaccessible: • 2a.1 The System presents a error message, saying that the database is inaccessible • 2a.2 The System does not add the occurrence of plans on the metrics plot
Minimal guarantee:	The plot presents, at least, the metrics values
Success guarantee:	The System presents to the Administrator the metric values and the plans that took place in the defined time interval

A.4.8 Use Case - Export Chart

Table A.29: Use Case 4.8 - Export Chart

Name:	Export Chart
Primary Actor:	Administrator
Priority:	Must
Stakeholders and Interests:	Administrator wants to export charts containing metrics values and their associated plans for posterior analysis.
Preconditions:	System triggered Use Case - Visualize resource metrics
Main Success scenario:	1. Administrator selects the option to export the chart to pdf 2. The system converts the chart into a pdf file and saves it on Administrator's device
Extensions:	2a. There is an error while converting/saving the image: • 2a.1 The use case ends
Minimal guarantee:	Chart can not be exported
Success guarantee:	The System converts the metrics and plans chart into a pdf file and saves it on Administrator's device

A.4.9 Use Case - Associate Configuration Profile to Resource

Table A.30: Use Case 4.9 - Associate Configuration Profile to Resource

Name:	Associate Configuration Profile to Resource
Primary Actor:	Administrator
Priority:	Could
Stakeholders and Interests:	Administrator wants to define the configuration profile that is going to be used for a resource
Preconditions:	• Administrator has, at least, one configuration profile • System triggered Use Case - Create Resource or Update Resource

Main Success scenario:	1. System presents the list of configuration profiles the Administrator has 2. Administrator selects a configuration profile 3. System triggers Use Case - Preview metrics tree with the root metric associated to the selected configuration profile 4. Administrator confirms the selection 5. System associates selected configuration profile to the resource
Extensions:	1a. Database is inaccessible: • 1a.1 The System presents a message, saying that the list of configuration profiles of the user could not be retrieved because database is inaccessible • 1a.2 System redirects Administrator to the homepage 4a. Administrator wants to change its configuration profile selection: • 4a.1 Administrator selects another configuration profile • 4a.2 Use case continues at step 3 5a. Database is inaccessible: • 5a.1 The System presents a message, saying that the configuration profiles could not be associated to the resource because database is inaccessible • 5a.2 System redirects Administrator to the homepage
Minimal guarantee:	No configuration profile can be associated to a resource
Success guarantee:	Selected resource has a configuration profile associated

A.4.10 Use Case - Export Chart Configuration

Table A.31: Use Case 4.10 - Export Chart Configuration

Name:	Export Chart Configuration
Primary Actor:	Administrator
Priority:	Must
Stakeholders and Interests:	Administrator wants to export the configuration of a chart to later add it on the homepage. Thus, he can easily monitor a system and also, directly and manually change some parameters to produce other charts of interest.
Preconditions:	System triggered Use Case - Visualize resource metrics
Main Success scenario:	1. Administrator selects the option to export the chart configuration 2. The system generates a readable configuration file with the options introduced to generate the current chart
Extensions:	
Minimal guarantee:	The chart configuration file is generated.
Success guarantee:	The same as the Minimal guarantee.

A.5 Probes

A.5.1 Use Case - View Probe Information

Table A.32: Use Case 5.1 - View Probe Information

Name:	View Probe Information
Primary Actor:	Administrator
Priority:	Could
Stakeholders and Interests:	Administrator wants to view Probe's Information
Preconditions:	Administrator navigates on the dashboard to the Probe's section and selects the option to see detailed information.

Main Success scenario:	1. The system presents a list of Probes 2. Administrator selects a Probe 3. The System presents information on the id, name, password, token and token expiration of the selected Probe 4. **Extension Points:** Use Case - Update Probe/ Use Case - Delete Probe
Extensions:	1a. Database is inaccessible: • 1a.1 The System presents a message, saying that the probes could not be gathered because database is inaccessible • 1a.2 The System redirects Administrator to homepage
Minimal guarantee:	Probe's information can not be seen
Success guarantee:	The information of the selected probe is shown

A.5.2 Use Case - Create probe

Table A.33: Use Case 5.2 - Create probe

Name:	Create probe
Primary Actor:	Administrator
Priority:	Could
Stakeholders and Interests:	Administrator wants to register a new developed probe in TMA
Preconditions:	Administrator navigates on the dashboard to the probes' section and selects the option to create

Main Success scenario:	1. System presents a form with name, password, token and token expiration fields to be filled 2. Administrator fills the form 3. System verifies that the form was well filled 4. System creates probe with given name, password, token and token expiration 5. System presents a message saying that the probe was created 6. System redirects the Administrator to the homepage
Extensions:	3a. The Administrator incorrectly fills in the Probe parameters: • 3a.1 The system presents information about the error nature • 3a.2 The use case continues at step 2 4a. Database is inaccessible: • 4a.1 The System presents a message, saying that the probe could not be created because database is inaccessible • 4a.2 The use case continues at step 6
Minimal guarantee:	There is no change in the database when it comes to Probes
Success guarantee:	The register of a new probe is accomplished and the database is updated accordingly

A.5.3 Use Case - Update Probe

Table A.34: Use Case 5.3 - Update Probe

Name:	Update Probe
Primary Actor:	Administrator
Priority:	Won’t
Stakeholders and Interests:	Administrator wants to update a Probe’s Information
Preconditions:	System triggered Use Case - View Probe Information

Main Success scenario:	1. Administrator selects the option to edit 2. The System presents editable information on the name, password, token and token expiration of the selected probe 3. Administrator updates probe's information 4. System verifies that the parameters are well filled 5. System updates the probe's information 6. System presents a message, saying the probe was updated 7. System redirects the Administrator to the home-page
Extensions:	4a. Some parameter is not correctly filled: • 4a.1 The system verifies that a parameter was badly defined • 4a.2 The system presents a message, saying that a parameter was badly defined, along with information on why • 4a.3 The use continues at step 3 5a. Database is inaccessible: • 5a.1 The System presents a message, saying that the probe could not be updated because database is inaccessible • 5a.2 The use case continues at step 7
Minimal guarantee:	There is no change on the selected probe
Success guarantee:	The selected probe is updated

A.5.4 Use Case - Delete probe

Table A.35: Use Case 5.4 - Delete probe

Name:	Delete probe
Primary Actor:	Administrator

Priority:	Won't
Stakeholders and Interests:	Administrator wants to delete a probe previously registered in TMA.
Preconditions:	System triggered Use Case - View Probe Information
Main Success scenario:	1. Administrator chooses the option to delete 2. System deletes selected probe 3. System presents a message saying that the probe was deleted 4. System redirects Administrator to the homepage
Extensions:	2a. Database is inaccessible: • 2a.1 The System presents a message, saying that the probe could not be deleted because database is inaccessible • 2a.2 The use case continues at step 4
Minimal guarantee:	None probe is deleted from TMA's database
Success guarantee:	The deletion of a probe is accomplished and the database is updated accordingly

A.6 Actuators

A.6.1 Use Case - View Actuator Information

Table A.36: Use Case 6.1 - View Actuator Information

Name:	View Actuator Information
Primary Actor:	Administrator
Priority:	Could
Stakeholders and Interests:	Administrator wants to view Actuator's information
Preconditions:	Administrator navigates on the dashboard to the Actuator's section and selects the option to see detailed information.

Main Success scenario:	1. The system presents a list of Actuators 2. Administrator selects an Actuator 3. The System presents information on the address and public key of the selected Actuator 4. **Extension Points:** Use Case - Update Actuator / Use Case - Delete actuator
Extensions:	1a. Database is inaccessible: • 1a.1 The System presents a message, saying that the actuators could not be gathered because database is inaccessible • 1a.2 The System redirects Administrator to home-page
Minimal guarantee:	Actuator's information can not be seen
Success guarantee:	The information of the selected actuator is shown

A.6.2 Use Case - Create actuator

Table A.37: Use Case 6.2 - Create actuator

Name:	Create actuator
Primary Actor:	Administrator
Priority:	Could
Stakeholders and Interests:	Administrator wants to register a new developed actuator in TMA.
Preconditions:	Administrator navigates on the dashboard to the actuator's section and selects the option to register.

Main Success scenario:	1. System presents a form with address and public key fields to be filled 2. Administrator fills the form 3. System verifies that the form was well filled 4. System creates actuator 5. System presents a message saying that the actuator was created 6. System redirects Administrator to homepage
Extensions:	3a. The Administrator fills in incorrectly some parameter: • 3a.1 The system presents information about the error nature • 3a.2 The use case continues at step 2 5a. Database is inaccessible: • 5a.1 The System presents a message, saying that the actuator could not be created because database is inaccessible • 5a.2 The use case continues at step 6
Minimal guarantee:	None actuator is registered in TMA's database
Success guarantee:	The register of a new actuator is accomplished and the database is updated accordingly

A.6.3 Use Case - Update Actuator

Table A.38: Use Case 6.3 - Update Actuator

Name:	Update Actuator
Primary Actor:	Administrator
Priority:	Won't
Stakeholders and Interests:	Administrator wants to update an Actuator's Information
Preconditions:	System triggered Use Case - View Actuator Information

Main Success scenario:	1. Administrator selects the option to edit 2. The System presents editable information on the address and public key of the selected actuator 3. Administrator updates actuator's information 4. System verifies that the parameters are well filled 5. System updates actuator's information 6. The system presents a message, saying the actuator was updated 7. The system redirects the Administrator to the homepage
Extensions:	4a. Some parameter is not correctly filled: • 4a.1 The system verifies that a parameter was badly defined • 4a.2 The system presents a message, saying that a parameter was badly defined, along with information on why • 4a.3 The use continues at step 3 5a. Database is inaccessible: • 5a.1 The System presents a message, saying that the actuator could not be updated because database is inaccessible • 5a.2 The use case continues at step 7
Minimal guarantee:	There is no change on the selected actuator
Success guarantee:	The selected actuator is updated

A.6.4 Use Case - Delete actuator

Table A.39: Use Case 6.4 - Delete actuator

Name:	Delete actuator
Primary Actor:	Administrator
Priority:	Won't

Stakeholders and Interests:	Administrator wants to delete an actuator previously registered in TMA.
Preconditions:	System triggered Use Case - View Actuator Information
Main Success scenario:	1. Administrator chooses the option to delete 2. The system deletes selected actuator 3. The System presents a message saying that the actuator was deleted 4. The system redirects Administrator to the homepage
Extensions:	2a. Database is inaccessible: • 2a.1 The System presents a error message, saying that the actuator could not be deleted because database is inaccessible • 2a.2 The use case continues at step 4
Minimal guarantee:	None actuator is deleted from TMA's database
Success guarantee:	The deletion of an actuator is accomplished and the database is updated accordingly

A.7 Actions

A.7.1 Use Case - View Action information

Table A.40: Use Case 7.1 - View Action information

Name:	View Action information
Primary Actor:	Administrator
Priority:	Could
Stakeholders and Interests:	Administrator wants to view an Action's information
Preconditions:	Administrator navigates on the dashboard to the Action's section and selects the option to view information

Main Success scenario:	1. The System presents a list containing actions' ids and names 2. Administrator selects an Action 3. System displays id and name of the selected Action, a list of configurations with their ids, keyname and domain values, and associated actuator's id, actuators' address, resource's id and resource's name 4. **Extension Points:** Use Case - Update Action / Use Case - Delete Action
Extensions:	1a / 3a. Database is inaccessible: • 1a.1 / 3a.1 The System presents a message, saying that the database is inaccessible • 1a.2 / 3a.2 System redirects Administrator to the homepage
Minimal guarantee:	None kind of information can be seen
Success guarantee:	Detailed information on an Action can be seen

A.7.2 Use Case - Create Action

Table A.41: Use Case 7.2 - Create Action

Name:	Create Action
Primary Actor:	Administrator
Priority:	Could
Stakeholders and Interests:	Administrator wants to add an action to TMA
Preconditions:	Administrator navigates on the dashboard to the Action's section and selects the option to create

Main Success scenario:	1. System presents a form with name field and actuator and resource associations 2. Administrator fills in the name 3. System triggers Use Case - Associate Resource 4. System triggers Use Case - Associate Actuator 5. System verifies that the form and associations were well done 6. System persists action 7. System presents a message saying that the action was created 8. System redirects Administrator to homepage
Extensions:	5a. Some parameter is not correctly filled: • 5a.1 The system verifies that a parameter was badly defined • 5a.2 The system presents a message, saying that a parameter was badly defined, along with information on why • 5a.3 The use continues at step 2 6a. The database is inaccessible: • 6a.1 The System presents a message, saying that the action could not be created because database is inaccessible • 6a.2 Use case continues at step 9
Minimal guarantee:	TMA's database remains the same when it comes to actions
Success guarantee:	A new Action is added to the TMA's database along with its resource and actuator associations

A.7.3 Use Case - Update Action

Table A.42: Use Case 7.3 - Update Action

Name:	Update Action

Primary Actor:	Administrator
Priority:	Won't
Stakeholders and Interests:	Administrator wants to update the information of an action
Preconditions:	System triggered Use Case - View Action Information
Main Success scenario:	1. Administrator selects the option to edit 2. The System presents editable information on the name of the selected Action 3. **Extension Points:** Use Case - Create Configuration / Use Case - Delete Configuration 4. Administrator updates Action's information 5. System verifies that all of action's information were well set 6. The System updates the action 7. System presents a message saying that the action was updated 8. System redirects Administrator to the homepage
Extensions:	5a. Some parameter is not correctly filled: • 5a.1 The system verifies that a parameter was badly defined • 5a.2 The system presents a message, saying that a parameter was badly defined, along with information on why • 5a.3 The use continues at step 3 6a. TMA's database is inaccessible: • 6a.1 The System presents a message, saying that the action could not be updated because database is inaccessible • 6a.2 Use case continues at step 7
Minimal guarantee:	Selected action's information remains the same as the beginning of the use case
Success guarantee:	The selected action's information is updated

A.7.4 Use Case - Delete Action

Table A.43: Use Case 7.4 - Delete Action

Name:	Delete action
Primary Actor:	Administrator
Priority:	Won't
Stakeholders and Interests:	Administrator wants to delete an action
Preconditions:	• Action to be deleted can not be being used on adaptation rules by TMA's Planning • System triggered Use Case - View Action Information
Main Success scenario:	1. Administrator selects the option to delete 2. System deletes selected action along with its configurations and plans' associated data 3. System presents a message saying that the action and any related data were deleted 4. System redirects Administrator to the homepage
Extensions:	2a. TMA's database is inaccessible: • 2a.1 The System presents a message, saying that the action and any related data could not be deleted because database is inaccessible • 2a.2 Use case continues at step 4
Minimal guarantee:	None action or associated data are deleted from TMA's database
Success guarantee:	The selected action, its configurations and associated data are deleted from TMA's database

A.7.5 Use Case - Create Configuration

Table A.44: Use Case 7.5 - Create Configuration

Name:	Create Configuration
Primary Actor:	Administrator

Priority:	Could
Stakeholders and Interests:	Administrator wants to add configurations for an action
Preconditions:	Administrator is updating an action
Main Success scenario:	1. Administrator selects the option to add configurations for the current action 2. System presents a empty list of configurations for the current action and an option for creating a configuration 3. Administrator selects the option to create a configuration 4. System presents a form with keyname and domain fields 5. Administrator fills the form 6. System verifies that the form was well filled 7. System creates configuration for the action 8. System presents a message saying that the configuration was created and added to the action's list 9. Administrator navigates back to the screen where the action can be updated

Extensions:	2a. Configurations may have been added for the action before: • 2a.1 System presents a list with the existing configurations for the action and an option for creating a new configuration on this list • 2a.2 Use case continues at step 3 6a. Some parameter is not correctly filled: • 6a.1 The system verifies that a parameter was badly defined • 6a.2 The system presents a message, saying that a parameter was badly defined, along with information on why • 6a.3 The use continues at step 5 7a. The database is inaccessible: • 7a.1 The System presents a message, saying that the configuration could not be created because database is inaccessible • 7a.2 System redirects Administrator to the homepage 9a. Administrator wants to keep adding configurations for the current action: • 9a.1 Use case continues at step 2
Minimal guarantee:	Selected Action has no changes on the configurations available for it
Success guarantee:	Configurations are created for the selected action

A.7.6 Use Case - Delete Configuration

Table A.45: Use Case 7.6 - Delete Configuration

Name:	Delete Configuration
Primary Actor:	Administrator
Priority:	Could
Stakeholders and Interests:	Administrator wants to delete a configuration for an action

Preconditions:	• Administrator is updating an action • There is at least one configuration for the current action • Configuration to be deleted can not be being used on adaptation rules by TMA's Planning
Main Success scenario:	1. Administrator selects the option to delete configurations for the current action 2. System presents a list of configurations for the current action and an option for deleting a configuration 3. Administrator selects the option to delete a configuration 4. System deletes the configuration for the current action and any plan's associated data 5. System presents a message saying that the configuration for the current action was deleted 6. Administrator navigates back to the screen where the action can be updated
Extensions:	4a. TMA's database is inaccessible: • 4a.1 The System presents a message, saying that the configurations and associated plan's data could not be deleted because database is inaccessible • 4a.2 System redirects Administrator to the homepage 6a. Administrator wants to keep deleting configurations for the current action: • 6a.1 Use case continues at step 3
Minimal guarantee:	Configuration and plan's associated data will always be both deleted, they can never be deleted apart.
Success guarantee:	Configurations for an action and associated plan's data are deleted.

A.7.7 Use Case - Associate Resource to Action

Table A.46: Use Case 7.7 - Associate Resource to Action

Name:	Associate Resource to Action
Primary Actor:	Administrator
Priority:	Could
Stakeholders and Interests:	Administrator wants to associate an action to a resource
Preconditions:	Administrator is creating an action
Main Success scenario:	1. System presents a list of resources containing name and id 2. Administrator selects a resource 3. System associates the action to the selected a resource at the frontend
Extensions:	1a. TMA's database is inaccessible: • 1a.1 The System presents a message, saying that the resources could not be gathered because database is inaccessible • 1a.2 System redirects Administrator to the homepage
Minimal guarantee:	No association can be done between the currently being created action and a resource.
Success guarantee:	The action being created is associated to a resource at the frontend

A.7.8 Use Case - Associate Actuator to Action

Table A.47: Use Case 7.8 - Associate Actuator to Action

Name:	Associate Actuator to Action
Primary Actor:	Administrator
Priority:	Could
Stakeholders and Interests:	Administrator wants to associate an action to an actuator
Preconditions:	Administrator is creating an action

Main Success scenario:	1. System presents a list of actuators containing their addresses and ids 2. Administrator selects an actuator 3. System associates the action to the actuator at the frontend
Extensions:	1a. TMA's database is inaccessible: • 1a.1 The System presents a message, saying that the actuators could not be gathered because database is inaccessible • 1a.2 System redirects Administrator to the homepage
Minimal guarantee:	No association can be done between the currently being created action and an actuator.
Success guarantee:	The action being created is associated to an actuator at the frontend

A.8 Adaptation Rules

A.8.1 Use Case - View Adaptation Rule Detail

Table A.48: Use Case 8.1 - View Adaptation Rule Detail

Name:	View Adaptation Rule Detail
Primary Actor:	Administrator
Priority:	Must
Stakeholders and Interests:	Administrator wants to view information on an adaptation rule
Preconditions:	• TMA's Planning is running • Administrator navigates on the dashboard to the Adaptation Rule's section and selects the option to view a rule's information

Main Success scenario:	1. The System presents to the Administrator the list of adaptation rules names used by TMA's planning component 2. Administrator selects a rule 3. System displays condition of adaptation and the adaptation plan of the selected rule 4. **Extension Points:** Use Case - Update Adaptation Rule / Use Case - Delete Adaptation Rule
Extensions:	1a. TMA's planning is unreachable: • 1a.1 The system presents a message, saying TMA's Planning is unreachable • 1a.2 The Administrator is redirected to the homepage
Minimal guarantee:	None kind of information can be seen
Success guarantee:	Adaptation condition and plan of a Rule are presented

A.8.2 Use Case - Create Adaptation Rule

Table A.49: Use Case 8.2 - Create Adaptation Rule

Name:	Create Adaptation Rule
Primary Actor:	Administrator
Priority:	Must
Stakeholders and Interests:	Administrator wants to add an adaptation rule to TMA's Planning
Preconditions:	• TMA's Planning is running • Administrator navigates on the dashboard to the Adaptation Rule's section and selects the option to create

Main Success scenario:	1. The System loads information from actions and their configurations and presents a screen for an adaptation rule creation. 2. Administrator writes a name for the rule, sets the condition that triggers the adaptation, and iteratively chooses, by order of execution, the actions and their configurations to form the adaptation plan, filling in the values of actions' configurations. 3. System verifies that all parameters were well set 4. The system adds the new adaptation rule to the set held by TMA's Planning component 5. The System presents a message saying that the rule was added to TMA-Planning's set of adaptation rules 6. System redirects Administrator to the homepage

Extensions:	1a. The database is inaccessible and information can not be retrieved: • 1a.1 The System presents a message, saying that the actions and configurations could not be retrieved because database is inaccessible • 1a.2 Use case continues at step 6 3a. Some parameter is not correctly filled: • 3a.1 The system verifies that a parameter was badly defined • 3a.2 The system presents a message, saying that a parameter was badly defined, along with information on why • 3a.3 The use continues at step 2 4a. TMA's planning is unreachable: • 4a.1 The system presents a message, saying that the rule could not be added to the set of adaptation rules held by TMA's Planning because the component is unreachable • 4a.2 The use continues at step 6
Minimal guarantee:	TMA's Planning set of Rule's remains the same
Success guarantee:	A new adaptation rule is added to the set held by TMA's Planning component

A.8.3 Use Case - Update Adaptation Rule

Table A.50: Use Case 8.3 - Update Adaptation Rule

Name:	Update Adaptation Rule
Primary Actor:	Administrator
Priority:	Won't
Stakeholders and Interests:	Administrator wants to update the information of an adaptation rule
Preconditions:	• TMA's Planning is running • System triggered Use Case - View Adaptation Rule Detail

Main Success scenario:	1. Administrator selects the option to edit 2. The System presents editable information on the name and configuration values of the selected adaptation rule 3. Administrator updates adaptation rule's information 4. System verifies that all of adaptation rule's information were well set 5. The System updates the adaptation rule's information on TMA's Planning 6. System presents a message to the Administrator saying that the adaptation rule was updated 7. System redirects Administrator to the homepage
Extensions:	4a. Some parameter is not correctly filled: • 4a.1 The system verifies that a parameter was badly defined • 4a.2 The system presents a message, saying that a parameter was badly defined, along with information on why • 4a.3 The use continues at step 3 5a. TMA's planning is unreachable: • 5a.1 The system presents a message, saying that the adaptation rule could not be updated because TMA's Planning is unreachable • 5a.2 The use continues at step 7
Minimal guarantee:	Selected adaptation rule's information remains the same as the beginning of the use case
Success guarantee:	TMA-P updates the information of the edited adaptation rule

A.8.4 Use Case - Delete Adaptation Rule

Table A.51: Use Case 8.4 - Delete Adaptation Rule

Name:	Delete Adaptation Rule
Primary Actor:	Administrator
Priority:	Must
Stakeholders and Interests:	Administrator wants to delete an adaptation rule on TMA-P
Preconditions:	• TMA's Planning is running • System triggered Use Case - View Adaptation Rule Detail
Main Success scenario:	1. Administrator selects the option to delete 2. System deletes selected adaptation rule from the existing set of TMA's Planning component 3. System presents a message saying that the adaptation rule was deleted from the set held by TMA-Planning 4. System redirects Administrator to the homepage
Extensions:	2a. TMA's planning is unreachable: • 2a.1 The system presents a message, saying that the adaptation rule could not be deleted from the set held by TMA Planning because it is unreachable • 2a.2 The use continues at step 4
Minimal guarantee:	TMA's Planning set of adaptation rules remains the same
Success guarantee:	The selected adaptation rule is deleted from the set held by TMA's Planning

A.9 Plans

A.9.1 Use Case - List occurred plans

Table A.52: Use Case 9.1 - List occurred plans

Name:	List occurred plans

Primary Actor:	Administrator
Priority:	Should
Stakeholders and Interests:	Administrator wants to visualize a brief list with little information on the adaptation plans that have been made
Preconditions:	• A plan took place • Administrator navigates on the dashboard to the Plan's section
Main Success scenario:	1. Administrator selects the option to view occurred plans 2. The system loads the list of all plans that were done 3. The system presents the list of plans with information about its id and timestamp 4. **Extension Point:** Use Case - View occurred plan detail
Extensions:	2a. Administrator knows the id or timestamp of a plan: • 2a.1 The Administrator adds filters on the occurred plans, such as id or time they took place • 2a.2 The system loads from TMA's database the list of plans that match the filters added by the Administrator • 2a.3 Use case continues at step 3 2b. / 2a.2a TMA's database is inaccessible: • 2b.1 / 2a.2a.1 The System presents a error message, saying that the database is inaccessible • 2b.2 / / 2a.2a.2 The System redirects Administrator to the homepage
Minimal guarantee:	None information is given on occurred plans
Success guarantee:	The System, respecting Administrator's filters, presents a list of plans that have been made along with a brief information on their ids and time of occurrence

A.9.2 Use Case - View occurred plan detail

Table A.53: Use Case 9.2 - View occurred plan detail

Name:	View occurred plan detail
Primary Actor:	Administrator
Priority:	Should
Stakeholders and Interests:	Administrator wants to visualize, in detail, what actions and configurations were applied in an occurred plan
Preconditions:	System triggered Use Case - List occurred plans
Main Success scenario:	1. Administrator selects one of the plans on the list 2. The system presents a detailed information on the selected plan such as: id, timestamp, actions taken, actions' configurations and values used, as well as order of actions execution
Extensions:	2a. TMA's database is inaccessible: • 2a.1 The System presents a message, saying that the plan's detailed information could not be retrieved because database is inaccessible • 2a.2 The System redirects the Administrator to the homepage
Minimal guarantee:	Administrator is redirected to the homepage in case detailed information can not be retrieved from the database
Success guarantee:	The System presents to the Administrator the detail of an occurred plan, including actions taken and their configuration values

A.10 Logs

A.10.1 Use Case - Visualize logs

Table A.54: Use Case 10.1 - Visualize logs

Name:	Visualize logs
Primary Actor:	Administrator
Priority:	Should

Stakeholders and Interests:	Administrator wants to visualize the logs TMA produced in order to gather information on the management of the platform and monitored resources
Preconditions:	• TMA produced logs • Administrator navigates on the dashboard to the Logs section
Main Success scenario:	1. Administrator selects the option to view logs 2. The system presents the logs 3. **Extension Point:** Use Case - View specific logs
Extensions:	2a. TMA's database is inaccessible: • 2a.1 The System presents a message, saying that the logs could not be retrieved because database is inaccessible • 2a.2 The system redirects the Administrator to the homepage
Minimal guarantee:	None logs are presented
Success guarantee:	All logs are presented

A.10.2 Use Case - View specific logs

Table A.55: Use Case 10.2 - View specific logs

Name:	View specific logs
Primary Actor:	Administrator
Priority:	Should
Stakeholders and Interests:	Administrator wants to visualize specific logs TMA produced in order to gather a more detailed information on the management of the platform and monitored resources
Preconditions:	• TMA produced logs • System triggered Use case - Visualize logs

Main Success scenario:	1. Administrator applies filters on the logs at time/description/component/origin 2. The system presents the logs filtered
Extensions:	2a. TMA's database is inaccessible: • 2a.1 The System presents a message, saying that the logs could not be filtered because database is inaccessible • 2a.2 The system redirects the Administrator to the homepage
Minimal guarantee:	Logs can not be seen
Success guarantee:	Logs are presented and they match the filters applied

A.10.3 Use Case - Delete logs

Table A.56: Use Case 10.3 - Delete logs

Name:	Delete logs
Primary Actor:	Administrator
Priority:	Won't
Stakeholders and Interests:	Administrator wants to delete logs from TMA in order to save some space as they are not needed anymore
Preconditions:	• TMA produced logs • Administrator navigates on the dashboard to the Logs section

Main Success scenario:	1. Administrator selects the option to delete 2. System provides a form where starting and end dates must be filled 3. Administrator fills the start and end dates 4. The system deletes the logs between the start and end dates defined 5. The system presents a message saying that the logs were deleted 6. The system redirects the Administrator to the homepage
Extensions:	4a. TMA's database is inaccessible: • 4a.1 The System presents a message, saying that the logs could not be deleted because database is inaccessible • 4a.2 The use case continues at step 6
Minimal guarantee:	No logs are deleted from TMA's database
Success guarantee:	The logs created between the start and end dates specified are deleted

A.11 Dashboard

A.11.1 Use Case - Add chart to Homepage

Table A.57: Use Case 11.1 - Add chart to Homepage

Name:	Add chart to Homepage
Primary Actor:	Administrator
Priority:	Must
Stakeholders and Interests:	Administrator wants to import a chart config file to the homepage so that he can directly and easily access charts from monitored resources.
Preconditions:	A valid chart configuration file exists

Main Success scenario:	1. Administrator selects the option to import the chart configuration 2. System asks for the selection of a file 3. Administrator selects a file 4. System reads the file contents and asks for a name to be associated with the chart 5. Administrator introduces a name for the chart and selects the option to save 6. System saves the read chart configuration and its associated name in the database and generates the chart
Extensions:	3a. Administrator wants to cancel the operation: • 3a.1 Administrator cancels the file selection • 3a.2 The use case ends 5a. Administrator does not fill the name: • 5a.1 System alerts for the mandatory insertion of a name for the chart • 5a.2 The use case continues at step 5 5b. Administrator introduces a name that is associated with another chart: • 5b.1 System alerts Administrator that the name introduced is being used with another chart and asks for another name • 5b.2 The use case continues at step 5 6a. The database is inaccessible: • 6a.1 The System presents a message, saying that the configuration could not be created because database is inaccessible • 6a.2 Use case continues at step 5
Minimal guarantee:	No chart is added to the homepage nor to the database

Success guarantee:	A chart is added to the homepage and, consequently, on the database.

A.11.2 Use Case - Replace chart on Homepage

Table A.58: Use Case 11.2 - Replace chart on Homepage

Name:	Replace chart on Homepage
Primary Actor:	Administrator
Priority:	Must
Stakeholders and Interests:	Administrator wants to replace a chart with another because the current one is no longer needed.
Preconditions:	• A valid chart configuration file exists • There is a chart at the homepage
Main Success scenario:	1. Administrator selects the option to replace the chart configuration 2. System asks for the selection of a file 3. Administrator selects a file 4. System reads the file contents and asks for a name to be associated with the chart 5. Administrator introduces a name for the chart and selects the option to replace 6. System replaces the old chart configuration and associated name by the new ones and generates the chart with the new configuration

Extensions:	3a. Administrator wants to cancel the operation: • 3a.1 Administrator cancels the file selection • 3a.2 The use case ends 5a. Administrator does not fill the name: • 5a.1 System alerts for the mandatory insertion of a name for the chart • 5a.2 The use case continues at step 5 5b. Administrator introduces a name that is associated with another chart or with the replaced one: • 5b.1 System alerts Administrator that the name introduced is being used with another chart and asks for another name • 5b.2 The use case continues at step 5 6a. The database is inaccessible: • 6a.1 The System presents a message, saying that the configuration could not be created because database is inaccessible • 6a.2 Use case continues at step 5
Minimal guarantee:	No chart is replaced in the homepage nor in the database
Success guarantee:	A chart is replaced in the homepage and, consequently, in the database.

A.11.3 Use Case - Delete chart on Homepage

Table A.59: Use Case 11.3 - Delete chart on Homepage

Name:	Delete chart on Homepage
Primary Actor:	Administrator
Priority:	Must
Stakeholders and Interests:	Administrator wants to delete a chart because it is no longer needed.
Preconditions:	There is a chart

Main Success scenario:	1. Administrator selects the option to delete the chart configuration 2. System deletes the chart configuration and associated name
Extensions:	2a. The database is inaccessible: • 2a.1 The System presents a message, saying that the configuration could not be deleted because database is inaccessible • 2a.2 Use case continues at step 5
Minimal guarantee:	No chart is replaced in the homepage nor in the database
Success guarantee:	A chart is replaced in the homepage and, consequently, in the database.