Universidade do Minho
Escola de Engenharia
Departamento de Informática

Pedro Miguel da Costa Capa

Data Center Visualization

February 2022

Universidade do Minho
Escola de Engenharia
Departamento de Informática

Pedro Miguel da Costa Capa

Data Center Visualization

Master dissertation
Integrated Master's in Informatics Engineering

Dissertation supervised by
José Orlando Pereira
António Ramires Fernandes

February 2022

ACKNOWLEDGEMENTS

I would like to thank my family for giving me during all these years their support and the conditions to accomplish my goals.

I also want to thank professor José Orlando Pereira, as well as António Ramires, for their advice, patience, and assistance on this dissertation.

I want to thank MACC and engineer Rui Ribeiro for spending time and resources in helping the development of the dissertation.

STATEMENT OF INTEGRITY

I hereby declare having conducted this academic work with integrity.

I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration.

I further declare that I have fully acknowledged the Code of Ethical Conduct of the University of Minho.

ABSTRACT

As more projects are financed by public institutions, demonstrating tasks and other information developed is increasingly required. Traditional monitoring tools like Prometheus are used in data centers to collect data from the machines and supervise them as the devices can malfunction and make the service unavailable. Currently, applications like Prometheus that display the machines' performance are limited to a restricted set of people: the data center managers. These applications have limited data visualization methods since their focus is on retrieving the data from the machines. These applications are associated with specialized frameworks in showing the machines' performance. These frameworks present the data in several visualization methods, such as graphic lines and gauge graphics. However, the forms of exposing data are not attractive for people in general. So, other ways to expose the data need to be developed.

Data visualization in three dimensions can expose data more attractively. Besides, 3D has some advantages over traditional ways. With a data friendlier exposition, catching the attention of people who do not manage the data center is easier.

This project aims to build an application to expose the data center's data in a 3D scenario. The data exposed are the machines' tasks, components, and performance. By exposing the data center's tasks and other information to the general public, the application can present to the viewer the usefulness of the data center. The application must have its components flexible, so any data center can use it. Moreover, those data centers should expose any visualization they desire through plugins.

To complete the goals, first, different techniques to explore and view the data are investigated. Several applications that expose data from a data center are analyzed to know the current status of these applications. Furthermore, different scenarios are constructed based on the research made. Using a tool capable of handling web requests makes the application available to everyone. Besides, the application is flexible in some parts of the architecture to be adaptable to any framework. So, any data center can use the application. Those parts are the server that contains the machines' performance data and the database management system. The system allows the creation of a plugin to communicate with the machine's performance server. Following a simple interface, a new plugin can be developed with relative ease. Besides, the webserver is replicable, making it adaptable to the data center's needs. Moreover, the application allows the creation of arbitrary 3D scenarios. By following a set of steps a simple 3D scenario can be built, including the visualization and communication server stages. Such a scenario can be expanded freely, as long as the communication API is observed. The created scenario functions as a plugin that can be inserted into the application effortlessly. The application's usefulness is validated through an experience with information from a real data center. Finally, the application's performance is corroborated, supporting a considerable amount of concurrent requests.

KEYWORDS Data center, Visualization, Performance, 3D, Flask, Three.js, Plugin.

RESUMO

Como há um número maior de projetos financiados por instituições públicas, é necessário revelar que tarefas e outras informações a instituição desenvolve. Ferramentas de monitorização como o Prometheus são usadas em centros de dados para recolher o desempenho das máquinas e supervisioná-las uma vez que podem ter problemas tornando o serviço indisponível. Aplicações como o Prometheus que exibem o desempenho das máquinas são limitados a um conjunto restrito de pessoas: os gestores dos centros de dados. Estas aplicações têm métodos de visualização limitados, uma vez que se focam em obter os dados das máquinas. Estas aplicações são associadas com aplicações especializadas em mostrar o desempenho das máquinas. Os dados são apresentados em vários métodos de visualização, como gráficos de linhas e de área. No entanto, estas formas não são atraentes para o público em geral. Portanto, é preciso usar outras formas de expor os dados.

Visualização de dados em três dimensões pode expor os dados de uma forma mais eficiente. Além disso, 3D tem algumas vantagens em relação às formas tradicionais. Com um cenário mais amigável, é mais fácil captar a atenção das pessoas.

Este projeto tem o objetivo de construir uma aplicação para expor os dados do centro de dados em 3D. Os dados expostos são as tarefas, o desempenho e os componentes das máquinas. Ao expor as tarefas para o público em geral a aplicação pode apresentar a utilidade do centro de dados. A aplicação deverá ter os componentes flexíveis para que qualquer centro de dados o possa usar. Além disso, os centros de dados deverão expor qualquer tipo de visualização que desejarem.

Para completar os objetivos, são investigadas diferentes técnicas de exposição de dados. São analisadas várias aplicações que expõem os dados de um centro de dados para conhecer o estado atual das mesmas. Além do mais, são construídos vários cenários com base nos dados da investigação. Ao usar uma ferramenta capaz de lidar com pedidos web torna-a disponível para todos. A aplicação também deve ser flexível em alguns dos componentes para serem adaptados a qualquer ferramenta. Desta forma qualquer centro de dados pode usar a aplicação. As partes flexíveis devem ser o servidor que contém os dados do desempenho das máquinas e a base de dados. O sistema permite o uso de diferentes plugins para comunicar com esse servidor. Ao seguir um conjunto de passos a criação do plugin é relativamente fácil. O servidor aplicacional é replicável, tornando o sistema adaptável para as necessidades do centro de dados. A aplicação permite o desenvolvimento de novos cenários 3D. Ao seguir um conjunto de passos é criado um cenário 3D simples, incluindo os passos da visualização e comunicação com o servidor. O cenário pode ser expandido, desde que siga a API de comunicação. O cenário criado funciona como um plugin que pode ser adicionado na aplicação facilmente. A utilidade da aplicação é validada através de uma experiência com dados reais de um centro de dados. Por fim, o desempenho da máquina é validado, uma vez que suporta uma quantidade considerável de pedidos concorrentes.

PALAVRAS-CHAVE Centro de dados, Visualização, Desempenho, 3D, Flask, Three.js, Plugin.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF LISTINGS

1

INTRODUCTION

The chapter starts by addressing the motivations for this project. Next, the problems and challenges of this project are illustrated. After that, the objectives of the dissertation are explained. Finally, an overview of the entire dissertation is made.

1.1 MOTIVATION

Many advancing computing centers are public institutions funded by public funds, so showing people what the data center does and manages within the organization is logical. Traditional monitoring tools like Prometheus are used in data centers to collect the machines' performance data and supervise them as the devices can malfunction and make the service unavailable. However, Prometheus is not the most appropriate application to display data because of its limits regarding graphic visualization, as it provides only a single line chart at each time. The visualization of metrics collected by these frameworks can be improved, by associating the application with other technologies to observe dashboards, like Grafana, which has better forms to visualize data. However, these frameworks are only available for system operators and administrators.

Furthermore, the current technologies are evolving and allow programmers to explore other techniques difficult to develop a few years ago. Exploring other forms of visualizing data beyond the traditional ways as three-dimensional scenarios is now more accessible in terms of framework and hardware availability. Creating an application that displays the data center in a user-friendly way and is available to everyone would help in the transparency of the data centers. With a friendly scenario, people may be interested in observing and learning about data centers.

Besides, with the pandemic, people's lives have changed, namely doing more work from home. The application allows people to visit the data center without leaving their houses.

1.2 PROBLEM

A large part of the data center's visualizations forms is oriented to system administrators, making them unattractive for regular people. Adding to the fact they do not know how a data center works, it makes them go unrecognized, despite being a big part of our lives. Creating an easy-to-understand scenario can be a good starting point to attract people to gain interest in data centers and what surrounds one.

In addition, regular applications are only available to data center managers, making it difficult for people to access data center information. Creating a web application makes the information available to everyone. A web application is more complex than an administrator application since the server's scalability has to be taken into account. The best way to make the application scalable is by replicating some of its components. The replication may involve some challenges. Besides, the applications that possess the information about the data center machines, such as Prometheus, are not prepared to receive hundreds of concurrent requests, so it must be found an efficient way to manage the data center's data.

Furthermore, the visualization offered by those applications is somewhat limited. If the data center managers want to expose the data in different forms, the application is not capable of it. The main challenge of the front-end is to find the most suitable way to expose the data center's information. Either with their geographical positions or in a completely different form, for instance, using a 3D scenario. However, creating a 3D world, even with friendly frameworks, can have its difficulties. Finding the best aesthetic way to expose the data in the scenario has its challenges. As people have different preferences, it will be hard to find the right way to expose the data so that it pleases everybody. Each created scenario also needs to take into account the dimension of the data center. As the dimension of the data center increases, so does the complexity of the scenario in computational processing. Providing a real-time navigation experience across a wide range of hardware, as is the case of web applications, is also a challenge. Therefore such a solution requires the rendering of the scenario to be the most efficient as possible.

1.3 OBJECTIVES

The objective of this dissertation is to create a web application to be used by any data center to expose its information to the general public. The application must be prepared to handle a significant workflow to support the clients' requests. Besides, the application's front-end must exhibit the machine's tasks, performance, and hardware components.

The application must be separated into several components to be easier to maintain, replicate and scale. The application must have efficient data management once the server that possesses the machines' performance information (Data Source) is not ready to receive hundreds of concurrent requests. Moreover, the application must be able to support any Data Source. The data center managers should be capable of adding a plugin to enable the communication between the application and the Data Source. The components that deal with hundreds of requests, such as the database and the web component, must be replicable to make the application scalable and robust.

To show the data to the client, several scenarios are created. One scenario exhibits a data center room similar to its real counterpart. A second scenario illustrates the data center information in a more metaphorical way. This latter scenario pretends to show that data center information can be displayed in a variety of ways. Besides, the application must allow the data center manager to create any other scenario that he sees fit. By following a simple set of steps, a simple scenario can be created, displaying metrics obtained from the data source.

The application must be tested with information from a real data center to validate its usefulness. Finally, performance was evaluated to verify if the application is capable of receiving hundreds of requests per second.

1.4 STRUCTURE OF THE DISSERTATION

This dissertation is separated into six chapters.

Chapter 2 provides a brief overview of the area of visualization. In the final part of Chapter 2, the roles of a data center visualization application are described, and different data visualization programs are examined.

Subsequently, Chapter 3 describes the application's architecture and the data present on the application's front-end.

In Chapter 4, the application's front-end and the algorithms used on the scenarios created are shown in detail.

Then, Chapter 5 describes the tools and algorithms used on the back-end and how the program works.

Next, in Chapter 6, the application evaluation is elaborated. More specifically, how the generic system described applies to the problem at hand is examined, and the performance tests realized to the application are analyzed.

Finally, in Chapter 7, the conclusions, the future work, and a summary of the work fulfilled in this project are presented.

2

STATE OF THE ART

In Section 2.1, the definition of data visualization is exposed. Later, in Section 2.2, the advantages and disadvantages of data visualization and how to improve it are evaluated. Next, in Section 2.3, the advantages and disadvantages of 3D data visualization and the comparisons between 2D and 3D graphics are evaluated. At the end of the chapter, in Section 2.4, the applications that display data from a data center are investigated.

2.1 WHAT IS DATA VISUALIZATION?

At some moment in our lives, everyone watched some form of data visualization. Even though it looks simple and easy to do, building a graphic can be complex.

Starting with a simple definition, data visualization is the graphic representation of data. *Edward Tufte* complements this definition in his book *The Visual Display of Quantitive Information* Tufte (2001). For him, data visualization is the exhibition of data through points, lines, a coordinated system, numbers, symbols, words, shading, and color.

Another different definition is given by *S. Card, J. MacKinlay, and B. Shneiderman* at Card et al. (1999). They define visualization as "the use of computer-supported, interactive, visual representations of data to amplify cognition", where cognition is the gain of knowledge. So, for them, data visualization is a way of representing information to explore the human cognitive system in recognizing images.

Two categories divide visualization as a research field according to Card et al. (1999). Scientific and Information visualization. Scientific visualization favors the physical world, like the human body or meteorology. Information visualization is the visual representation of ideas, abstract concepts, and relationships, like financial data or bibliographic sources. Software visualization is a specific field of information visualization defined as “a representation of computer programs, associated documentation data, that enhances, simplifies and clarifies the mental representation the software engineer has of the operation of a computer system” Mili and Steiner (2002).

Even though a graphical representation is usually better, however, a graph can not be alone. A plot should always be a part of something, so the context is relevant. The authors of Chen et al. (2008) recommend that graphical representations have a caption, a headline, and in some cases, an article to explain the data to the reader. A study, headed by *Fach* and *Strothotte*, theorized that using graphical connections between text and images can form links between visual and verbal associative memory formations, helping the reader understand and keep the information Fach and Strothotte (1994).

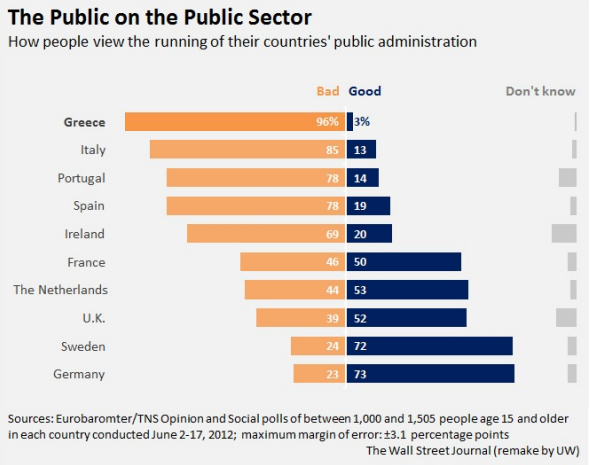

Figure 1: The people's opinion on the public sector in different countries. Source Inspari.

When constructing a graph is imperative to be meticulous with the caption's position, the scale, the colors, and the consistency between all the elements to ensure that the user can understand the displayed data. *Tufte* also refers in his book that data visualization is a recent specialty because it demands a few abilities. Although the construction of a graphic seems simple, *Tufte* thinks that most newspapers and magazines publish lying diagrams. In other words, they publish graphics so poorly constructed that the data represented does not describe the numerical information.

The exposition of data in graphics is more common than it looks. Data visualization is present in a few aspects of our lives, like the world map, the evolution of Covid-19 cases, or in most newspaper articles.

Figure 1 is an example of data visualization. The graphic exhibits the public opinion on the public sector on a horizontal graphic bar.

2.2 WHY USE DATA VISUALIZATION?

To comprehend why graphical representations are better the Human evolution is analyzed. After all, everything people can and cannot do is related to evolution.

Humans spend millions of years collecting food and running away from predators. So, improving the visual capacity to distinguish colors, forms, and other attributes was necessary. On the other hand, people perfected the ability to distinguish alphanumeric symbols for just a few millennia. So it does not have the same development compared to the ability to distinguish colors and forms. That is why people are not so efficient in watching data from letters compared to graphics. Even in some familiar situations, having a graphical representation is better than using letters. For example, if someone wants to discover a library on a map is easier to find it with a library graphical representation, e.g., an icon, compared with the letters LIBRARY on the map.

At this stage, visualization is a meaningful part of the Human cognitive system. In computational terms, vision is the highest bandwidth channel, inclusive, superior to all other senses combined Ware (2004). So, exploring

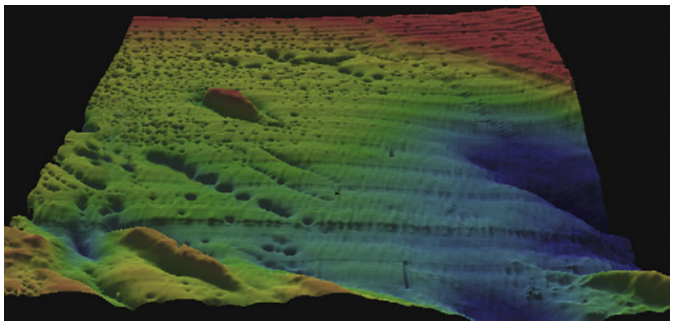

Figure 2: Passamaquoddy Bay. Bay localized between Maine in the U.S. and New Brunswick in Canada. Source Ware (2004).

the cognitive systems means an efficient search for data. Graphical representations are a powerful tool to help users to accomplish different types of mental processes. According to Teyseyre and Campo (2009).

- Exploratory. It happens when the user does not know what the goal is.
- Analytical. The user knows what to look for and is trying to determine its existence.
- Descriptive. The data is known, but the observer needs to have optical verification.

In addition, *Colin Ware* wrote at Ware (2004) a few reasons why visualization should be used.

- Visualization gives the capacity to comprehend a huge amount of data.
- The perception of properties and patterns.
- Reveals problems with the data.

In Figure 2, a 3D representation of the seabed marks is shown in lines and colors. Compared to a table-based solution, this approach allows the viewers to understand much better millions of measurements.

David McCandless, one expert in this area, gives another point of view. At David McCandless, *McCandless* reinforces some reasons presented previously on why people should make visual data. For him, data visualization is a way to compress data. *McCandless* gives an example of the number of breakups in a year, according to Facebook status. He compressed gigabytes of data from Facebook in a simple graphic. *McCandless* also affirmed that with a graphical representation is easier to find patterns. As seen in Figure 3, spring break is the period with more splits, and Christmas day is the day with fewer. The data are much easier to understand in a visual format than in lines of text and numbers.

A few authors evidence a tendency to use visual data in tasks, especially the more cognitive ones. For example, as *M.B. McGrath* and *J.R. Brown* wrote at McGrath and Brown (2005), scientists changed their mode of reasoning, because without visualization, scientists cannot do efficient examination Teyseyre and Campo (2009). With all the discussions they had with engineering educators, both conclude that visual thinking is crucial to the future of learning, and students should learn to create their visual data.

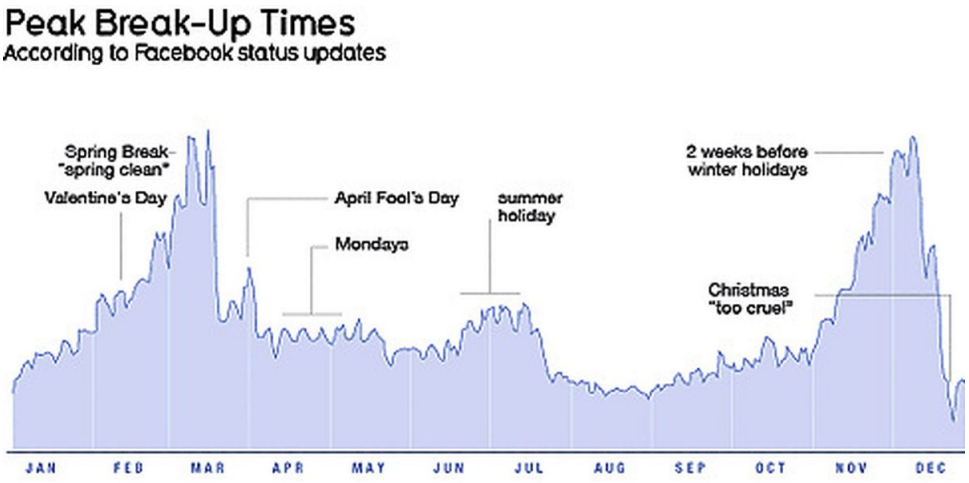

Figure 3: The number of breakups in 2008 according to Facebook status. Source Information is Beautiful.

According to *Tufte*, despite the use of graphs being a little disregarded in some areas, the idea that graphics are for unsophisticated people is becoming less popular. These are a few reasons why presenting the data using graphical representations to explore and show information is better.

To make a good data illustration, developing a graphic to show the data is not enough. For *Edward Tufte* excellence in graphics consists of deep ideas communicated with clearness and efficiency Tufte (2001). Thus, graphical displays should:

- show the data;
- induces the viewer to think about the data;
- make large data sets coherent;
- reveal the data at multiple levels;

To retrieve more advantages of visualization, the writer should use some tricks to improve the representation. *Colin Ware* said in his book that people have three simple high-level channels. Color, form, and motion Ware (2004). When a person sees an object, the recognition is a combination of simple visual attributes. Even though the object might take some time to identify, people comprehend the simple visual properties without effort. Because people make less effort to identify those simple visual attributes, the presenter should avoid complex objects and other elaborated optical details. *Stephen Few* shows in Figure 4 how an entity is praised by his characteristic at Few (2004).

So, in Figure 5 and Figure 6, how much time to find how many fives are in each figure is necessary? This example proves that using a different color for every number five (Figure 6) makes the task of counting the number of occurrences of this number much easier compared to a representation where the number five is not enhanced in some way (Figure 5).

When making a graphic representation, the most relevant information should have positive asymmetric preattentive alerts, Ware (2004), i.e., it should be easier to distinguish. For example, if the goal is to find a distinct

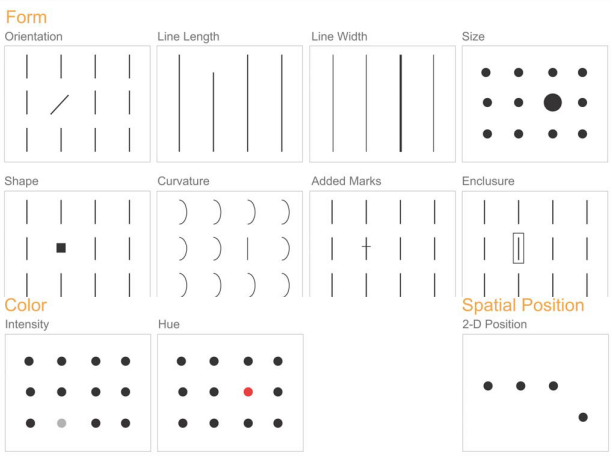

Figure 4: Visual attributes that exalt objects. Source Few (2004).

Figure 5: Number 5 without enhancement.

Figure 6: Number 5 with enhancement. Source Few (2004).

circle, a big one in the middle of hundreds of smaller circles is more flashy. This example is similar to colors, size, motion, forms, and other features.

Graphics can communicate data efficiently, but they can also be inadequate. As the audience easier remembers the visual illustration, an incorrect graphic effortlessly spoils the presentation. At this moment, with the technological evolution creating a graphic can be made instantly without any thought and seriousness Chen et al. (2008). Like any other part of any document, the writer must invest some time to develop the graphic.

Some factors affect the reaction time, like the visual noise volume and the distinction between the background elements. Those factors can increase exhaustion and diminish the capacity of visualization McGrath and Brown (2005). The ultimate goal of data visualization is optimizing the cognitive processes due to Human evolution. Specifically, data visualization uses visual data to communicate information in a universal aspect, quick and effective. The eyes are the principal source of information in our body, so exploring the best methods makes the process more efficient.

2.3 THREE DIMENSIONAL DATA VISUALIZATION

3D visualization has not been as studied as 2D. The hardware limitations and the complexity of creating them are a great reason. Besides, the first computers were focused on complex analytic calculations shifting the graphics building focus away Chen et al. (2008). Due to the hardware and software limitations, only in the last 20 years have graphics started to gain attention. Now, building new visualization forms, namely iterative 3D, is easier. However, the computer's power is limited. So, for now, exploring every technique is not possible.

Claus O. Wilke wrote in his book *Fundamentals of Data Visualization* when 3D graphics should be used and when should not Wilke (2019).

First, *Wilke* explains 3D plots problems. For the first reason, a 3D chart usually does not add any information, which means the 3D charts are not practical. Other authors, like *Tufte*, who says that graphical excellence is a graphic that gives to the viewer extensive ideas with the least ink agrees with *Wilke* Tufte (2001).

The second reason for *Wilke* is that sometimes the use of three-position scales can be hard to explain data. For *Wilke*, the writer can replace 3D graphics with 2D ones and replace the third dimension with scales, colors, size, or shapes.

The third reason is the conversion of 3D objects into two dimensions for showing the visualization on a monitor deforms the data. The human brain tries to fix the distortion by transforming the 2D print into a 3D image. However, the correction is only partial. *Wilke* gives an example of how a 3D graphic can distort data. Imagine a bar chart with two values, 75%, and 25%. The bar chart gives different perceptions consonant the inclination of the 3D graphic.

On the other hand, *Wilke* describes when 3D visualizations can be used. For him, 3D visualizations are suitable if the visualization is interactive so it has different perspectives of the scene, providing the necessary images for the human brain to reconstruct the scene. *Irani* and *Ware* at Irani and Ware (2004) and *Brath*, *Peters* and *Senior* at Brath et al. (2005), argue why 3D presentations are practical. For them, 3D presentations facilitate the perception of the human visual system. They believe that 3D graphics and animation can amplify

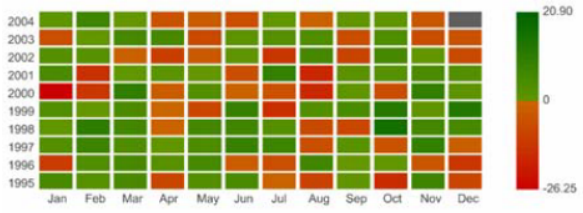

Figure 7: 2D representation of heat grid. Source Brath et al. (2005).

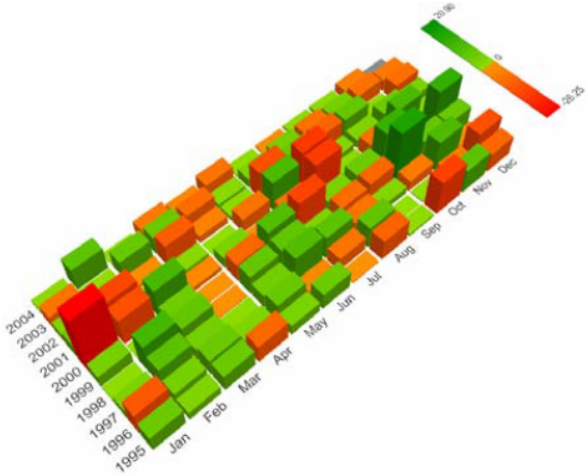

Figure 8: 3D representation of heat grid. Source Brath et al. (2005).

	Geon	UML
Novices	22.3	44.3
Experts	2.9	25.9
Globally	14.6	36.2

Table 1: Error percentage of Irani and Ware experience.

the visualization appeal, intuitiveness, and memorability. *Brath*, *Peters* and *Senior* wrote that when the goals of visualization includes communication the designer must include decorative details to enhance it Brath et al. (2005). As is presented in Figure 7 and Figure 8, substantial differences in colors are easy to distinguish in the 2D scenario. However, subtle differences are harder to judge. 3D makes subtle differences easier to distinguish.

Irani and *Ware* made an experience with several people who had to connect a text with a 2D UML diagram and a Geon diagram. A geon diagram is a 3D shaded solid composed of primitive shapes, like a cylinder. Figure 9 and Figure 10 shows an UML diagram and the equivalent geon diagram used on the experience, respectively. In the test, some people had experience in UML, and the others had not. Table 1 shows the results obtained from the *Irani* and *Ware* study Irani and Ware (2004), after the participants had a small training. They found out that users identify typical UML diagrams (2D) with higher error rates than the structures and relationship types of geon diagrams(3D).

On the second disadvantage refereed by *Wilke*, in which refers 3D is bad to represent data, Brath Brath et al. (2005) gave an example of why 3D graphics can be hard to interpret data, but it also showed that the appropriate

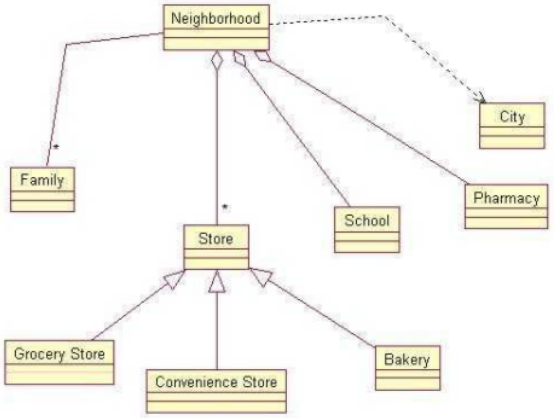

Figure 9: UML diagram used on the *Irani* and *Ware* experiences.

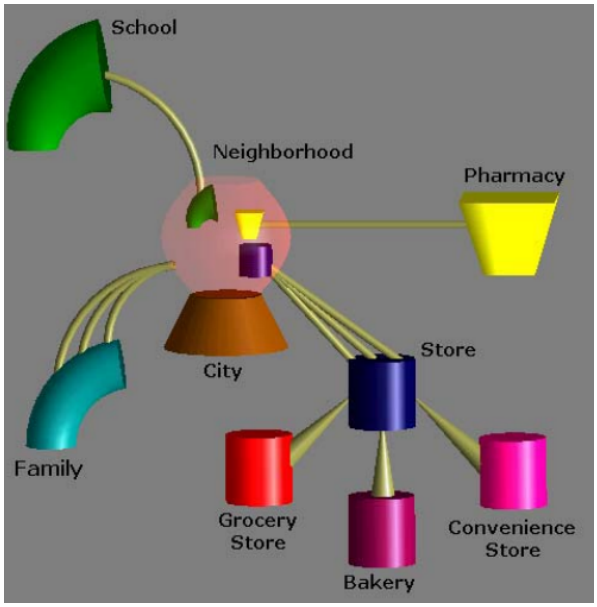

Figure 10: Geon diagram used on the *Irani* and *Ware* experiences.

configurations can be more effective than a 2D chart. Last, 3D is more natural because of the similarity with the world Teyseyre and Campo (2009).

To conclude, *Wilke* and *Tufte* consider that 3D should not be used because they can distort the data and do not bring any information, though other authors such as *Ware* and *Brath* think if the visualization is well built, the 3D graphics are better to understand the data.

2.4 DATA CENTER VISUALIZATION

The U.S. Environment Protection Agency defines a data center as "Primarily electronic equipment used for data processing (servers), data storage (storage equipment), and communications (network equipment). Collectively, this equipment processes, stores, and transmits digital information". And as "Specialized power conversion and backup equipment to maintain reliable, high-quality power, as well as environmental control equipment to maintain the proper temperature and humidity for the Information and Communication Technology (ICT) equipment" Geng (2014). Data centers are the places where a company's vital processes run. All the essential data from a business company is processed, stored, and organized in these places.

Data centers are taking part in people's activities, for example, Amazon, Facebook, Google, and Twitter. They are used directly or indirectly in foods, transportation, news, clouding, among other basic needs. Furthermore, the growth of data center usage also increases the power and money invested. So, the machines need monitoring to have the shortest downtime possible to avoid financially, client, and sometimes lives lost. The monitorization of the machines is achieved through applications that collect their performance. Since these applications are focused on registering the performance of the machines, they have limited data visualization methods. These applications are associated with other frameworks to display the machines' performance to data center managers.

Besides the private data centers that use machines with average processing (around three billion calculations per second), exist other data centers of High Processing Computing or Advanced Computing Center. These data centers perform complex tasks at high speed with machines that compute around quadrillions of calculations per second. One of the most known High Processing Computing solutions is the supercomputer. A supercomputer has several computer nodes that work together to complete tasks faster NetApp. This type of processing enables the tasking resolutions impossible to solve with standard computers. More specifically, the simulation of natural phenomena, supporting scientific advances such as new drugs, contribution to engineering tasks such as car design, and projects that handle huge volumes of data FCCN. These projects usually are suggested by public institutions such as universities that use these Advanced Computing Centers to develop the projects.

The next sub-chapters address three different applications that display the machine's performance.

2.4.1 *Grafana*

Grafana is an open-source visualization and analytics software written in Go and TypeScript released in 2014 by *Torkel Ödegaard*. Grafana Labs. Grafana allows the user to observe the machine's performance from numerous data sources and present it on personalizable diagrams for analysis. Grafana targeted time-series databases

Figure 11: Grafana dashboard.

such as InfluxDB and Prometheus but evolved to support relational databases such as MySQL and PostgreSQL. Each data source supported by Grafana has a particular query compiler for an optimal query work practice.

Grafana provides a huge variety of visualization options to help the administrator view and understand data. Grafana splits the visualizations into panels that compose its dashboard. A panel is a block in Grafana that displays the performance of a machine. The information pulled from the data source attributed to that panel can be displayed in several graph types (gauge, histogram, area chart, among others). For example, a panel could have a bar chart with the temperature of a machine over time. In Grafana, the user can personalize the type of graphics to use, the colors, sizes, metrics, and other elements. In addition, Grafana allows the user to import dashboards from other people. With that option, the user can import a page to visualize the data and retouch it. This option saves the time of building a new one. An example of a personalized dashboard that contains gauge and area charts is shown in Figure 11. That dashboard is imported and used as an example. The dashboard from Figure 11 only exposes the performance of a machine. Other dashboards, such as the dashboard of Figure 12 display multiple targets.

When monitoring applications, the manager needs to be aware of every malfunction to keep the systems healthy and reduce the machine's downtime. Grafana has a huge number of notification channels, such as email notifications. Besides, Grafana allows the managers to note on the panels providing a way to take notes on them. Usually, the notes are about actions, events, or ideas.

2.4.2 *Sunbird*

Sunbird has an option to visualize all the data in graphics, just like Grafana. Furthermore, it has another form to expose the data by making a geographic representation of the data center. In other words, they display the data in a three-dimensional scenario. One of the Sunbird goals is to make the data center visualization simple by removing the dependency on emails and spreadsheets to visualize the data center data in the same space Sunbird Inc.

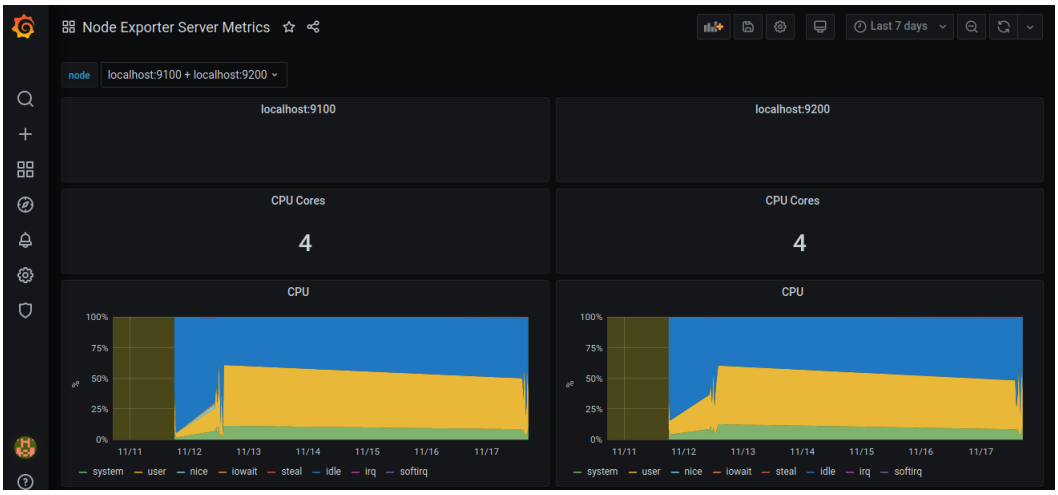

Figure 12: Grafana dashboard with multiple targets.

The Sunbird application has a default scenario for users to start testing functionalities. The 3D scene represents the room physically. It has a set of cabinets that are composed of a few compartments. Some of these compartments are unoccupied others have racks. Every cabinet has a color that represents the worst performance of the racks. As seen in Figure 13, three values are represented: the resource utility, weight, and potential power. The green cabinets mean that all the racks have parameters below the green limit. The yellow cabinets mean that all the racks have parameters below the yellow limit. The other cabinets have at least one parameter above the max yellow limit. All of the performance values limits are adjustable.

On the bottom left of Figure 13, the application exposes a table with the exact rack values. On the right side of the page, the popup displays the empty and used compartments of the cabinet. By pressing one of the racks on the right window, the user sees all the specifications of the chosen rack, as shown in Figure 14. The application also shows a physical image of the rack, as seen in Figure 15. The application also has some minor features, such as the data center displayed in two dimensions and the machines connected within the data center by cables.

The 3D visualization is friendly to users that do not manage the data center. Globally, Sunbird offers a 2D and a 3D visualization making the program versatile and usable in multiple situations. However, the 3D visualization limits the number of metrics visualized and does not display the global state of each machine.

2.4.3 *nuPSYS*

According to the website, nuPSYS is “a 3D-Visualization (physically and virtually) for data center infrastructure management (DCIM), uniquely capable of visualizing virtual machines, virtual network topology, physical rack space, physical device details, power utilization, and heat-map.” nuPSYS. So, in other words, nuPSYS is similar to the Sunbird 3D visualization because it draws the machines according to their geographic position. However, it has some differences. First, it does not have a global vision of the machines’ performance, as Figure 16 shows. To check the machine’s performance, the user needs to click on it.

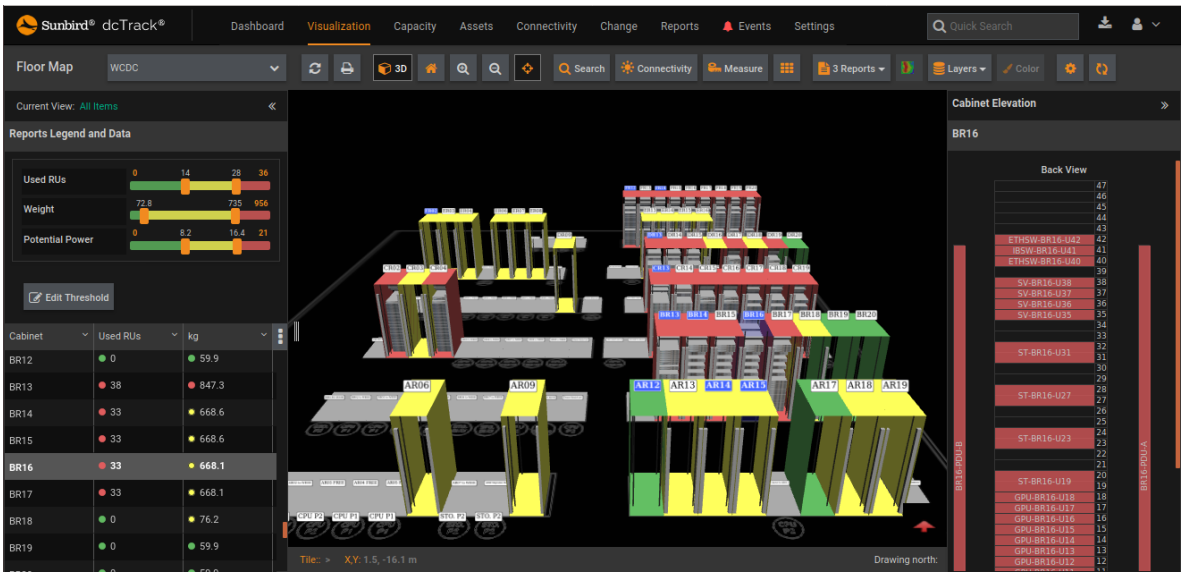

Figure 13: Sunbird data center scenario with the displayed cabinets' and machines.

An example of a machine's focusing vision is shown in Figure 18. On the rack window, the program exhibits the values of a machine with numbers and its limits through colors. If the user hovers over the device, it shows a popup with the machine's specification, such as the hardware model as shown in Figure 17.

The temperature in each part of the room can be exhibited on the floor, exposing a heat map, as seen in Figure 19.

nuPSYS decided to use visualization because the infrastructure complexity is growing, resulting in a challenge to identifying critical information effectively. According to them, visualization simplifies big data analytics.

2.4.4 *2D vs 3D applications*

Both types of data center visualization have pros and cons. The 2D visualization presents more precise data over time in a smaller space. For the data center managers, 2D is better because it has a significant amount of data compacted. Though, the 2D applications do not display the data center physically.

The 3D visualization applications display the data center machines physically closer to reality and are easier to observe the machine's general state. 3D visualization is more friendly for people who do not manage a data center. However, colors do not give exact values and, it can make a difference when managing the data center.

Combining the two types of data visualization can make the visualization perfect. Combining a 2D interface with numerical values and a 3D interface with a more general view of the data center takes advantage of both visualizations making the scenario more versatile.

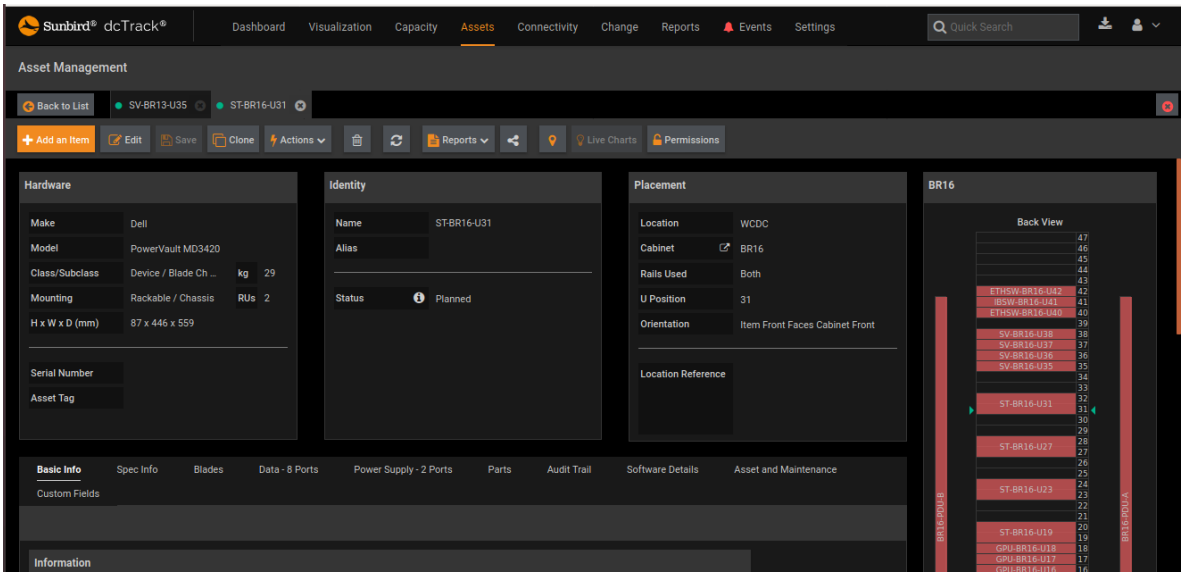

Figure 14: Panel with the rack specifications.

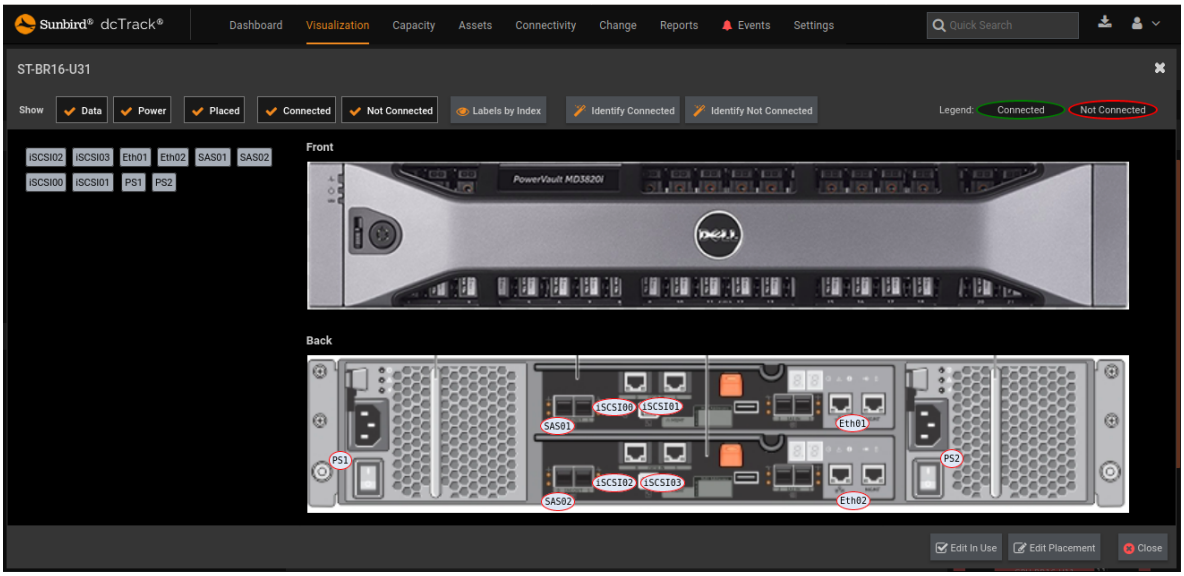

Figure 15: Panel with the machine's physical appearance.

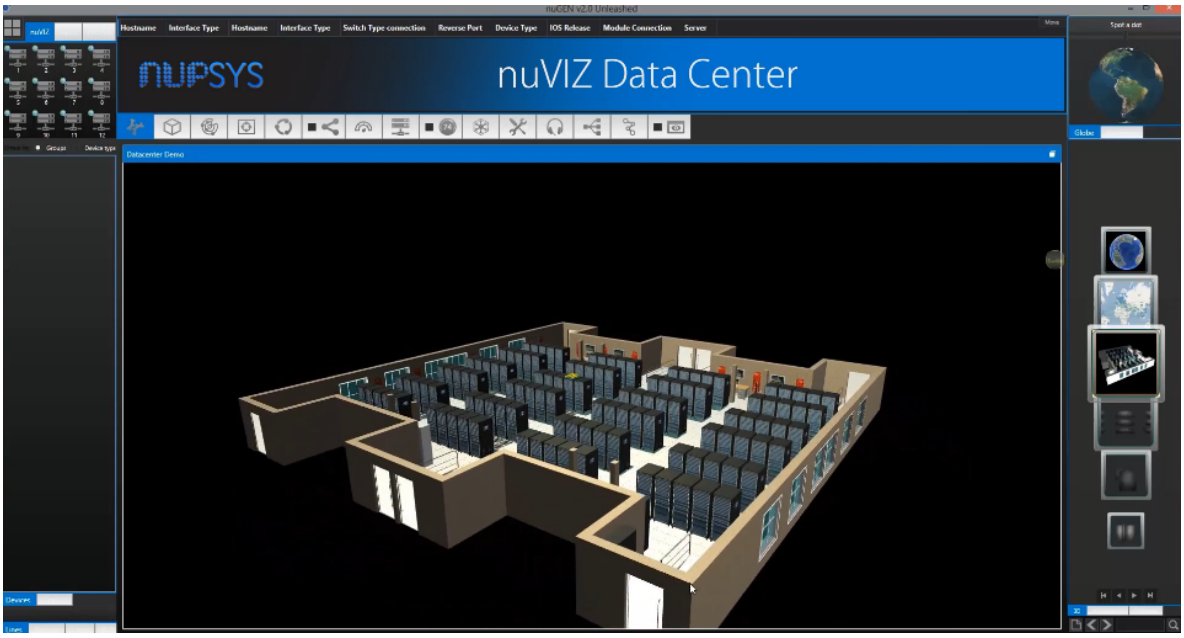

Figure 16: Global vision of data center on the nuPSYS application. Source nuPSYS.

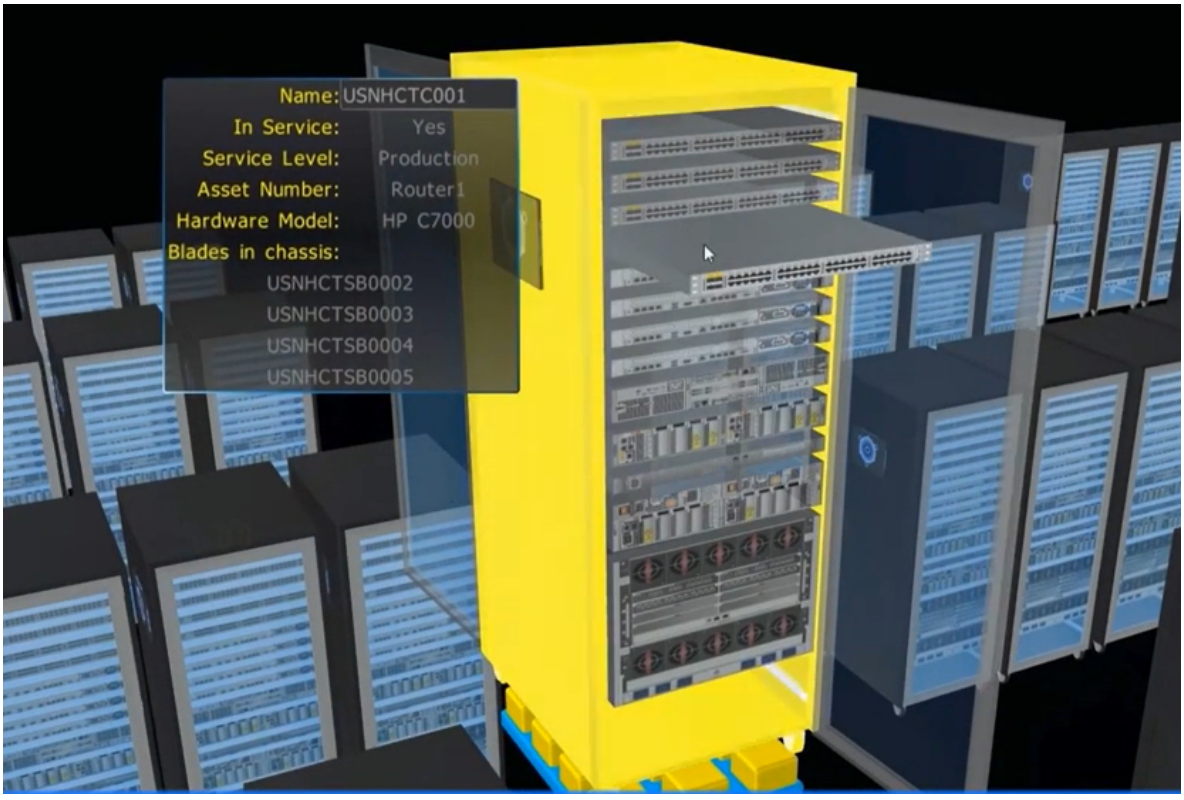

Figure 17: Popup with the machine's information. Source nuPSYS.

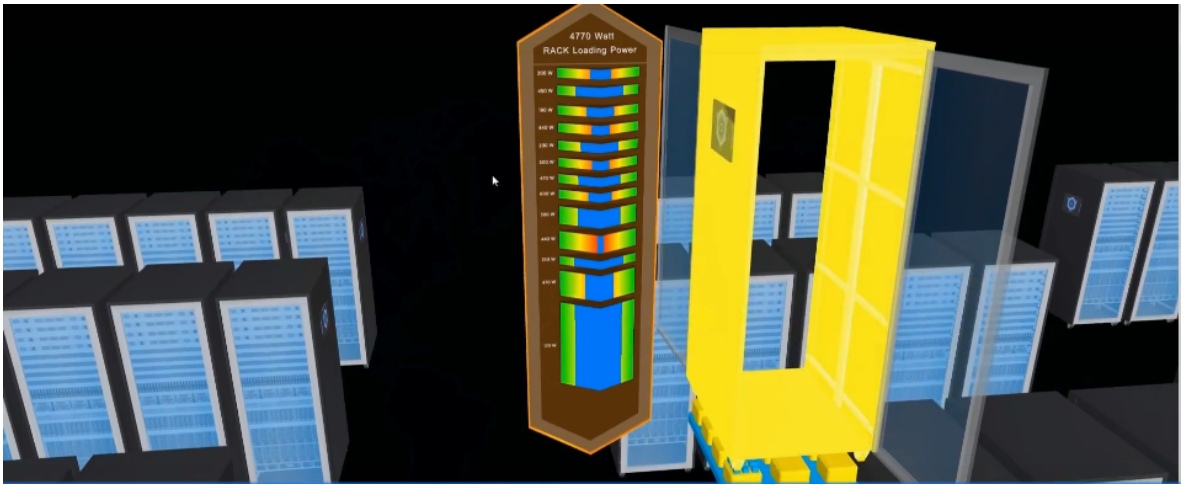

Figure 18: Cabinet close vision. Source nuPSYS.

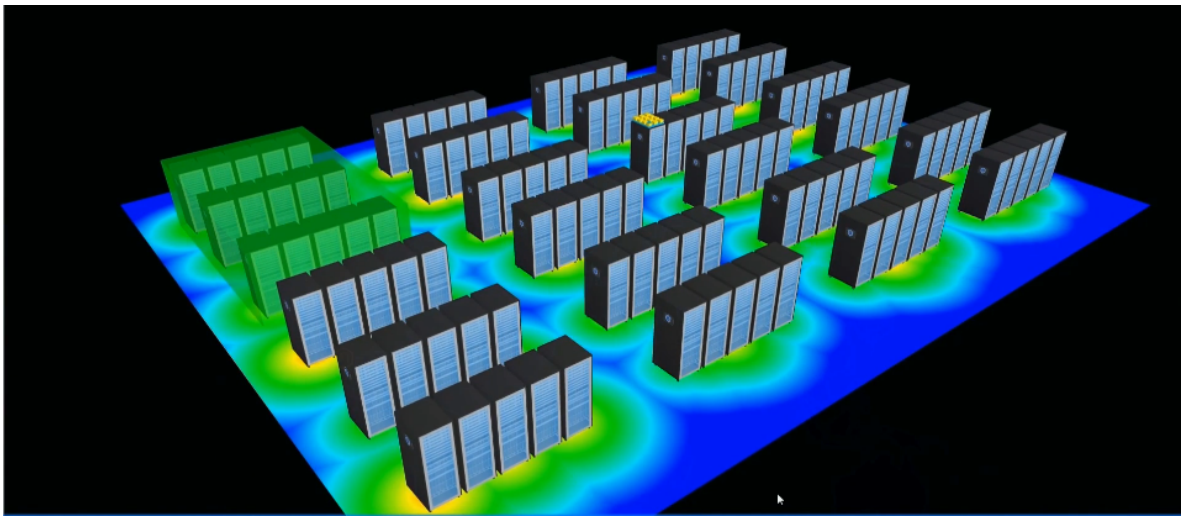

Figure 19: Floor heat map created with the room's temperature. Source nuPSYS.

2.4.5 *Critical analysis*

The applications described in this chapter are oriented toward data center administrators. These applications are great for observing the machine's information but are restricted to a few people once the data center managers are the application's users.

In another aspect, Grafana is flexible on the front-end of the application, where the administrator can edit the dashboards to multiple types of graphics, colors, sizes, and other characteristics. The other applications are not as flexible in this feature. However, the three applications have a common limitation. All three applications are relatively limited on what the manager can expose. If the visualization the user wants to display is not developed by the application, adding it is not possible.

3

SYSTEM DESIGN

Existing data center visualization applications have two main problems. They are only available to a restricted number of people, and the observable pages are somewhat limited in terms of what the program can offer. The proposed application does not limit the manager in the front-end views and is available on the web, potentially increasing its reach.

The data visualized on the front-end is presented in Section 3.1. Later, in Section 3.2, the architecture of the application is detailed.

3.1 DATA PRESENTATION

Appealing to users by representing the data center in a friendly scenario is the main objective of the application. The target audience is not data center managers but regular internet users. So the scenarios can not be too complex and expose too many alphanumeric symbols. Taking this into consideration, as an example, the application exposes the data center information in two distinct scenarios, a Room and a City.

The Room scene displays a data center room similar to its actual appearance. As the machines are a significant part of the data center, they are the main focus of the Room scenario. They are positioned according to their location on the data center. They are represented as a parallelepiped volume, similar to their form. Its location depends on the hall, cabinet, and drawer they belong to and the position within the drawer. Each of the hierarchies has an index that reveals its position within the room. Figure 20 and Figure 21 illustrates two diagrams of the data center room structure. The first diagram illustrates a hall composed of a set of cabinets. Each room can possess multiple halls. The second diagram indicates that each cabinet has a set of drawers, and these have a set of racks. From this predisposition, the scenario Room is created. This scenario contains several racks, also called machines, inside cabinets, as shown in Figure 22.

As mentioned before, one of the program goals is to show what the data center does. So, each machine is associated with a task that represents the current assignment. Furthermore, it has information regarding its Components. The Components are the machine's hardware pieces, such as RAM and CPU. Each Component has a set of Specifications. A Specification is the version of the Component. For example, for the RAM Component, the Specification could be a piece with 4 GB. Moreover, its performance is displayed as well through the color. The application is composed of a set of Metrics, which indicates how much a resource is being used or its condition. Each machine has an instant performance value and a range performance value per Metric. For example, Tem-

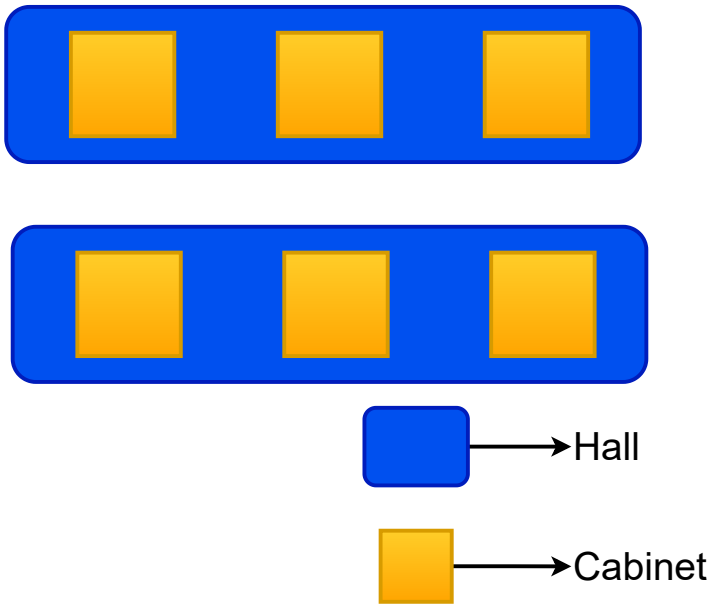

Figure 20: Room scenario from the birds eye perspective.

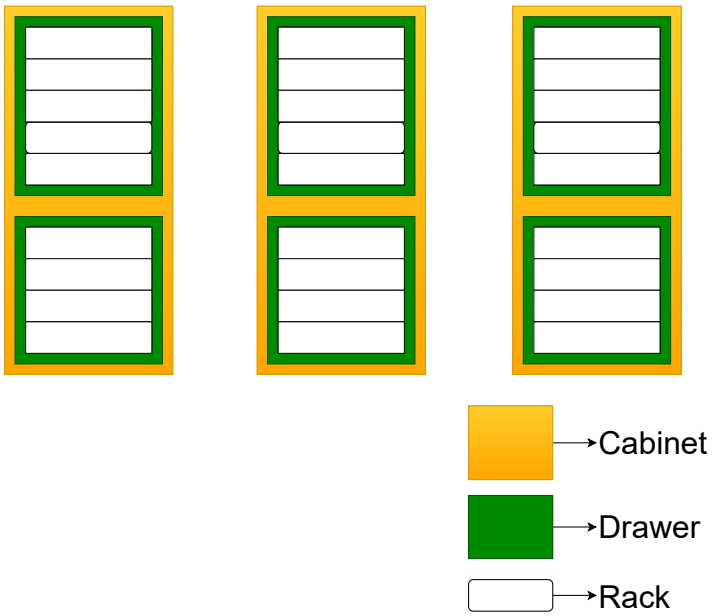

Figure 21: Room scenario cabinets shown with detail.

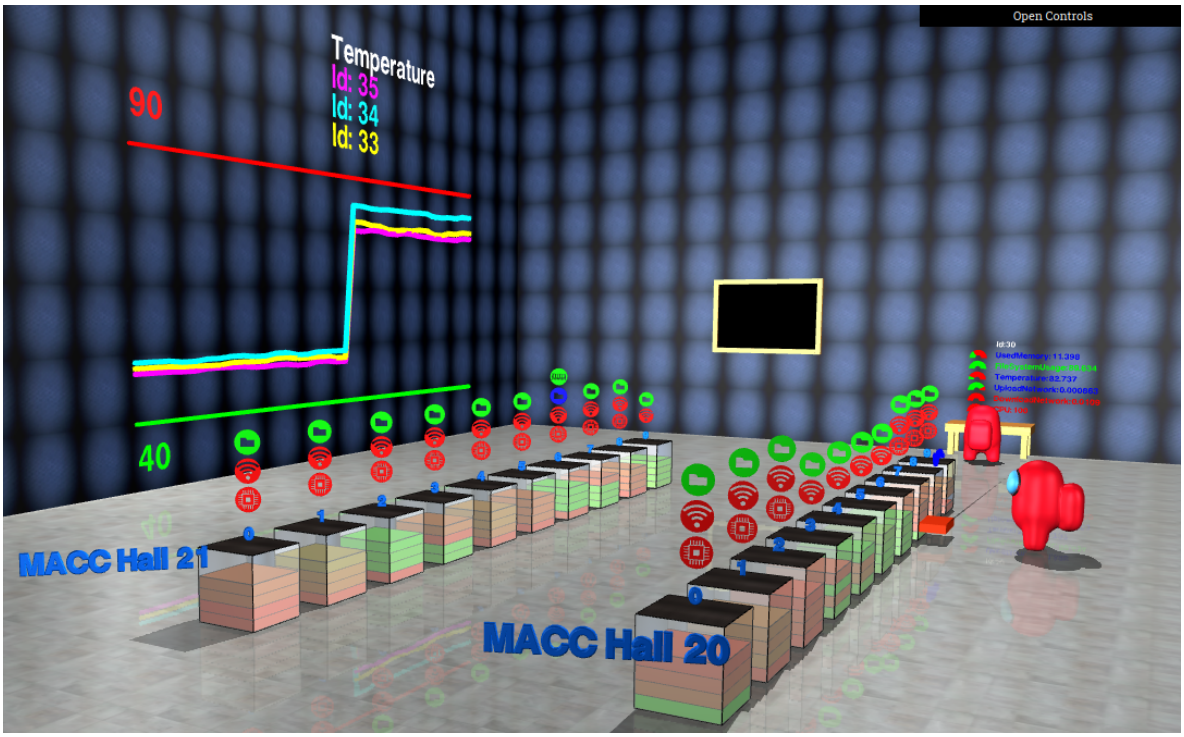

Figure 22: Room scenario view.

perature is a Metric. The machine's instant performance, or the machine's current performance, is a value that indicates how much a resource is being used or its condition. For example, the machine's current performance about the Temperature could be 40. Besides, the machine's range performance or machine's performance over time is a set of values that measures the performance of a Metric over a period.

Each Metric has two configuration values, namely, minimum and maximum. These variables indicate the limits where the performance is of concern, respectively. Within this scenario, the user can see the values of a specific machine by clicking on it.

By pressing it, the application exposes a popup with the machine's instant performance, components, and a task as shown in Figure 23. In addition, the application allows verifying its performance evolution by exposing the performance over time on the wall. The wall graphic follows the evolution of a specific Metric of several machines over time.

One last feature that the developed scenario exposes is the warning objects on the top of the cabinets. Whenever a machine is not within the Metric's thresholds, these warning signs become visible on the top of the cabinet to which belongs. Within these warning signs, an icon related to the metrics out of bounds is displayed. The color of the warning signs depends on whether the performances are below or over the Metrics thresholds. These elements in the warning signs help to detect the main performance problems of the machines in the cabinet.

The City scenario does not exhibit as much information about the data center as the Room scenario. This scenario does not have the purpose of showing all the data center's data. But to represent that the application can expose any imaginative scene from the data center information. The City scenario represents the machines

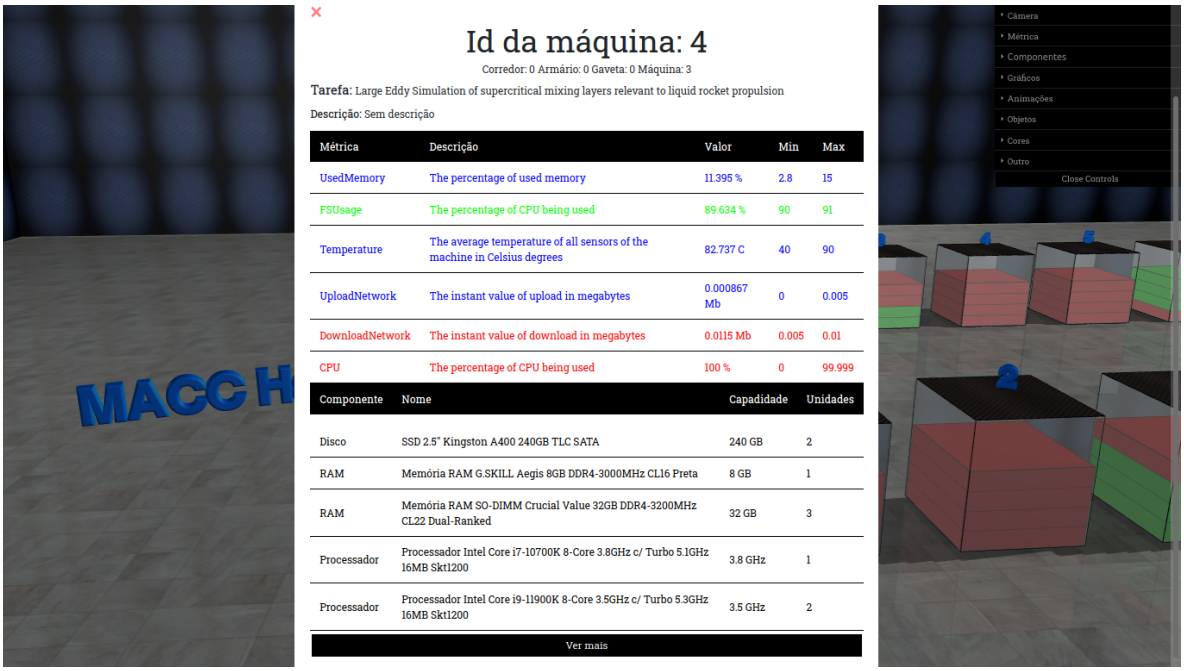

Figure 23: Popup with the performance, task, and components of a machine.

as houses, and their positioning depends on their location within the data center. In addition, the city scenario shows the data flow between machines of the data center by moving cars from and to the houses. Figure 24 illustrates part of the City's scenario, consisting of houses, walls, roads, trees, lamps, cars, and persons.

3.2 APPLICATION ARCHITECTURE

The system architecture allows the isolation of the different parts to easier maintenance. The implemented system is a web-based application that allows everyone to view the data center scenarios from any browser. Figure 25 exhibits the system organization and its components. The elements of the system are the Data Source, the Application Server and the Memcached, the Update Metrics Server, two databases, the Data Source Connection Plugin, and the Visualization Plugin.

First, the Data Source holds information about the data center machine's performance. Beyond that, the Application Server or Flask Server is an application that receives HTTP requests, processes them, and delivers the information to the clients to visualize the scenarios. This server is associated with a Memcached server. Memcached is an in-memory key-value store for small chunks of data that speeds up the requests. The way Flask and the system architecture are designed, allows it to have more than one Application Server. The visualization Plugin is a web page that displays a 3D scenario. So that the application can expose any scenario on the front-end about the data center, the system can have multiple Visualization Plugins. Furthermore, the Update Metrics Server requests the Data Source and stores the information on the database (Performance Values Database). A plugin on the Update Metrics Server is responsible for handling the request to the Data Source. Finally, the second database (Inventory database) saves the metadata about the data center, e.g., the machine's location.

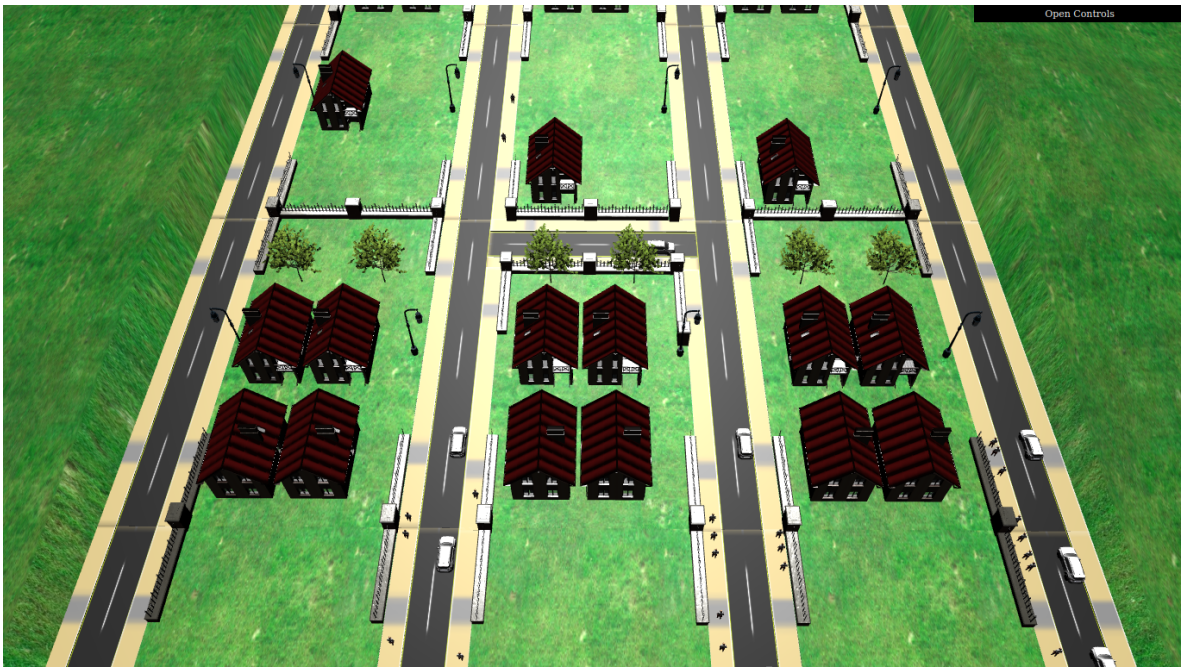

Figure 24: City scenario view.

To make the system flexible, any tool can be used as a Data Source. It can be an application like Prometheus, databases, or files. For this system, the Prometheus server is the default Data Source component. However, the application allows replacing Prometheus with another framework by replacing the Data Source Connection Plugin on the Update Metrics Server. For the plugin to retrieve data from a different Data Source, the application interface needs to be followed.

Just like the Data Source is replaceable, this application does not require a specific relational database management system. The application uses any relational database management system supported by SQLAlchemy. The SQLAlchemy job is explained later in this section.

The Data Source holds all the data about the machines' performance, so keeping a replica of the information on the Performance Values Database *a priory* would not make sense. Whenever a client requests the Application Server, the server could get the data directly from the Data Source. However, many applications that possess the machines' performance data are not prepared to receive hundreds of concurrent requests. The database, by storing those values reduces the number of requests from the application to the Data Source. Every time a client queries the Application Server, the information comes from the database or Memcached and not from the Data Source. This method eliminates the interaction between the Application Server and the Data Source. Furthermore, in terms of scalability is more simple to replicate a database than using multiple Data Sources.

Many frameworks available are capable of handling HTTP requests, like Django, ASP.NET, Ruby on Rails, and Flask. Flask is built in Python, has high compatibility with the latest technologies, is easy to develop and maintain, has a small core, easy to extend, and does not require specific tools. Considering those advantages, Flask is the framework used on this application. Flask has no database layer or other components but has several extensions,

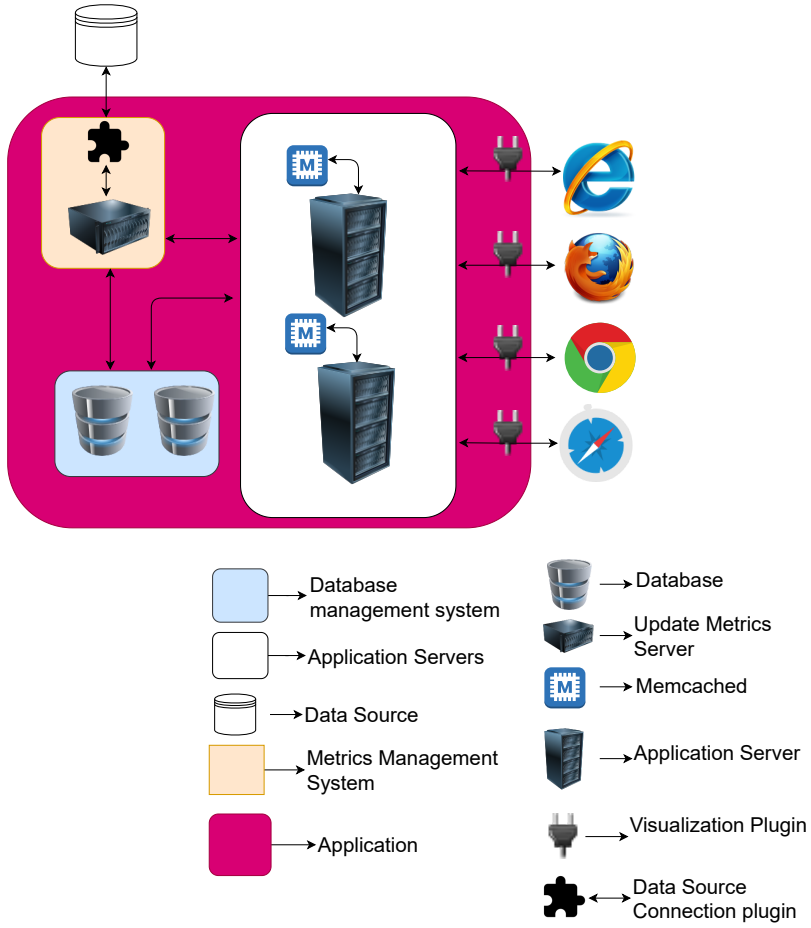

Figure 25: Application architecture.

facilitating the application development. One of the extensions available is the Jinja2 template engine. Regarding the tool used to interact with the database, SQLAlchemy is the chosen one.

Flask is fragmented into several parts: the routes, models, static files, and templates. The routes are the handlers that allow the users to make operations in their browsers, such as viewing an HTML file or asking for the machine's current performance. Moreover, the models are the classes used on the program and the database tables. Finally, the static and template files display the scenarios in the front-end.

To manage the data on the Performance Values Database, two strategies are available. The first alternative is making a separate server that periodically acquires the latest data and saves it on the database. The second option is to update the machine's performance values using the Application Server with a thread running in the background that handles the Data Source communication. The second technique would have the components on the same server. However, the deployment makes this alternative unfeasible. In the deployment, Flask uses Gunicorn (a Python Web Server Gateway Interface) to create several workers to handle the requests. Each worker creates a background thread and asks the Data Source periodically. This approach has several processes trying to update the machines' performance on the database when only one is needed. Other than that, Flask applications should only focus on receiving and responding to requests, and having a different thread running in the background would not make sense. Considering all the above, the first option was selected.

The design of this system requires the interaction between Application Server and Update Metrics Servers, namely for two specific cases. First, updating the machines' current performance values depends on the interaction between clients and the Application Server. If a data center Room had no visits in the past minutes, the Update Metrics Server stops updating the machines' performance values of that Room. Stopping requesting data reduces potential useless data exchanged between servers. The second situation is to ask the Data Source about the machines' performance over time. When a client requests the Application Server regrading the performance over a period, this checks its performance stored on the database. If some of the data is missing, the Application Server signals the Update Metrics Server to retrieve the data from the Data Source. When the Application Server requests the Update Metrics Server, the request message contains the message type and the machine identifier. And depending on the type of the request, it may contain a list of Metrics to require.

3.3 SUMMARY

In this chapter, the application's architecture and the scenarios developed for this project are exhibited. The Room scenario illustrates a room similar to the room's physical representation with the machines' performance, components, and assignments. In the City scenario, the program shows the data flow within the data center. Regarding the server architecture, this is composed of two databases, Application Server, Update Metrics Server, the Data Source, the Data Source Connection Plugin, and the Visualization Plugin. The Data Source and the scenarios are replaceable as long as the developed plugins follow a specific interface.

4

FRONT-END IMPLEMENTATION

The main purpose of the application is to expose the data in an animated scenario. For the data center information to be communicated effectively.

Section 4.1 describes a simple scenario that exposes data center information and is a base to other Visualization Plugins. As a Visualization Plugin example, the application shows two distinct scenarios. A Room is illustrated in Section 4.2, and a City is described in Section 4.3.

As this is a web-based application, it uses HTML, JavaScript, and CSS, namely the bootstrap library by Jacob Thornton. To build a 3-D scene in a web browser, several alternatives are available. Three.js is a high-level JavaScript based on WebGL that offers a set of predefined functionalities to help the production of the scenario. For example, this library has multiple object designs, like spheres, cylinders, alphanumeric symbols, and simple light and camera representations. Besides, Three.js is maintained, flexible, does not require plugins, is supported by any browser, and provides access to the power of the graphics of a computer.

To have a dynamic scene, Three.js has a function called *requestAnimationFrame*, which processes the logic of the scenario, such as changing the object's coordinates or color. Then, the program renders the frame according to all the positions, colors, and other attributes of the objects that belong to the scene and other scene properties such as lights and the camera. After rendering the scene, Three.js calls the function again to process the next frame. This function can be compared to an iterative function that only stops the recursion when the program ends.

In Three.js, every model is a JavaScript object defined by a material and a geometry. The material is the object's appearance, like the color, opacity, and in some cases, texture. The geometry is the object shape, such as vertices, edges, faces, and lines. A material and a geometry define a Mesh, which represents a triangular polygon mesh object.

4.1 VISUALIZATION PLUGIN

The visualization plugin allows the application's administrator to exhibit any scenario on the front-end. The visualization can be anything. Besides, the application is built to have multiple Visualization plugins.

To create a basic 3D world using Three.js, the Hello World Visualization plugin is presented. In this plugin, a box is drawn. The box depth depends on the machine's performance. Moreover, an example to obtain data from the server is described by requesting the machine's current performance.

```
let canvas = document.querySelector("#c");
let renderer = new THREE.WebGLRenderer({canvas, antialias: true});

let camera = new THREE.PerspectiveCamera(40, window.innerWidth / window.
    innerHeight, 0.1, 1200);
camera.position.set(-5, 5, 0);

let controls = new OrbitControls(camera, canvas);
controls.update();

let scene = new THREE.Scene();

let machine = 1;
getCurrentMetric(machine);

function render(time) {
    requestAnimationFrame(render);
    renderer.render(scene, camera);
}

requestAnimationFrame(render);
```

Listing 4.1: Simple Visualization Plugin of the JavaScript file.

The first step of the Visualization Plugin creation is the scenario. Listing 4.1 displays a part of the Hello World Visualization Plug-in JavaScript. To render the scene, the following elements must be defined. The WebGLRenderer displays the scene using WebGL. Next, the *camera* saves the state of the camera, such as its position. Moreover, the *controls* allow the user to navigate the scene using the mouse. Beyond that, the *scene* saves the state of the scenario, such as the object's state and lights. Finally, the *render* function renders the scene and creates new frames over time. The function also performs events. For example, defining a rotation of an object, the function performs the rotation in each frame.

So that the scenes can use data center information, it must make requests about the data to the server. Besides, these requests must be made in the background to not interrupt the animations. One of the most common solutions to get data in JavaScript is the AJAX library. AJAX is defined as Asynchronous JavaScript And XML. AJAX is a browser built-in XMLHttpRequest object to request data from a web server. AJAX allows web pages to be updated asynchronously by exchanging data with a web server in the background. The server request should be similar to the method on Listing 4.2. On the request, the identifier of the machine and the Metric are sent to a specific URL. In the response comes the Temperature machine's performance. When the JavaScript receives the response from the server, the plugin creates a BoxGeometry. This geometry defines the vertices and the planes to form a box with *width*, *height* and *depth*. The BoxGeometry depth depends on the machine's current performance. Three.js has more information on how to build a scenario using Three.js.

Finally, the HTML file that displays the web page is defined. Listing 4.3 exhibits the content of the HTML file. To run the 3D scenario, a few tags are necessary. The first script allows the web page to use the AJAX library. The second script indicates the JavaScript path defined previously. The canvas element is used to draw graphics on a web page. Figure 26 exhibits the Hello World Visualization Plugin scenario.

```
function getCurrentMetric(machine) {
    $.ajax({
        data: {
            rack: machine,
            metric: "Temperature"
        },
        type: "GET",
        url: "metrics"
    }).done(function(data) {
        if (data.error) {
            console.log(data.error);
        } else {
            let depth = JSON.parse(data);

            const geometry = new THREE.BoxGeometry(1, 1, depth);
            const material = new THREE.MeshPhongMaterial({color: 0x00ff00});
            const cube = new THREE.Mesh(geometry, material);
            scene.add(cube);

        }
    });
}
```

Listing 4.2: How to request a server using the AJAX library.

```
<!DOCTYPE html>
<html>
    <head>
    </head>
    <body>
        <canvas id="c"></canvas>
        <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.3/jquery.
            min.js"> </script>
        <script type="module" src="/static/js/helloworld.js"> </script>
    </body>
</html>
```

Listing 4.3: HTML file of the Hello World visualization Plugin.

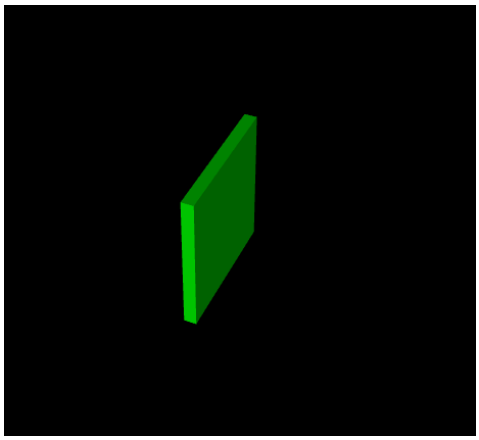

Figure 26: Hello World Visualization Plugin scenario.

4.2 ROOM

The Room Visualization plugin shows a scenario similar to a data center room, as shown in Figure 27. This scenario exhibits a set of objects that display the data center information. The colorful parallelepiped and the transparent boxes are the physical representation of the data center's machines and cabinets, respectively. Beyond that, the scenario has an avatar who diagnoses the machines and an observer who inspects the machine's performance the avatar is analyzing. These two objects make the scene more dynamic. Moreover, it has the cabinet's identification on top of them. At the left of the first cabinet on the hall, the plugin displays the hall's identity. Besides, at the top of each cabinet are displayed warnings signs to signal the user that a set of machines in the cabinet has performance issues. Finally, the scenario also has a graphic on the wall to show the machines' performance over time. To be easier for the user to interact with the scenario, Three.js usually has the library DAT.GUI associated with the scene Google Data Arts Team. For the plugin to display the entire scene, it must possess the machines' current performance and geographic location, the 2D warning shapes, tasks, components, and the 3D models.

4.2.1 *Machines*

As the project focuses on data center machines, these are the first objects to be detailed. One of the parts of the construction of the machine is its shape. As they have bodies similar to a parallelepiped, these objects are represented in the scene with a simple geometry by a BoxGeometry. Previously, in Chapter 3 is mentioned that the machine's geographic positions follow a structure, room, hall, cabinet, drawer, and rack. The machine's representation of the scenario follows that order. The machines' and cabinets' geographic position depends on the index of the structures they belong to. First, the machines that belong to a hall are on the same line. Second, machines that belong to the same cabinet have the same floor coordinates. The drawers that belong to the same cabinet are separated by a cylinder as shown in Figure 27. Besides, the machines and cabinets are attached to

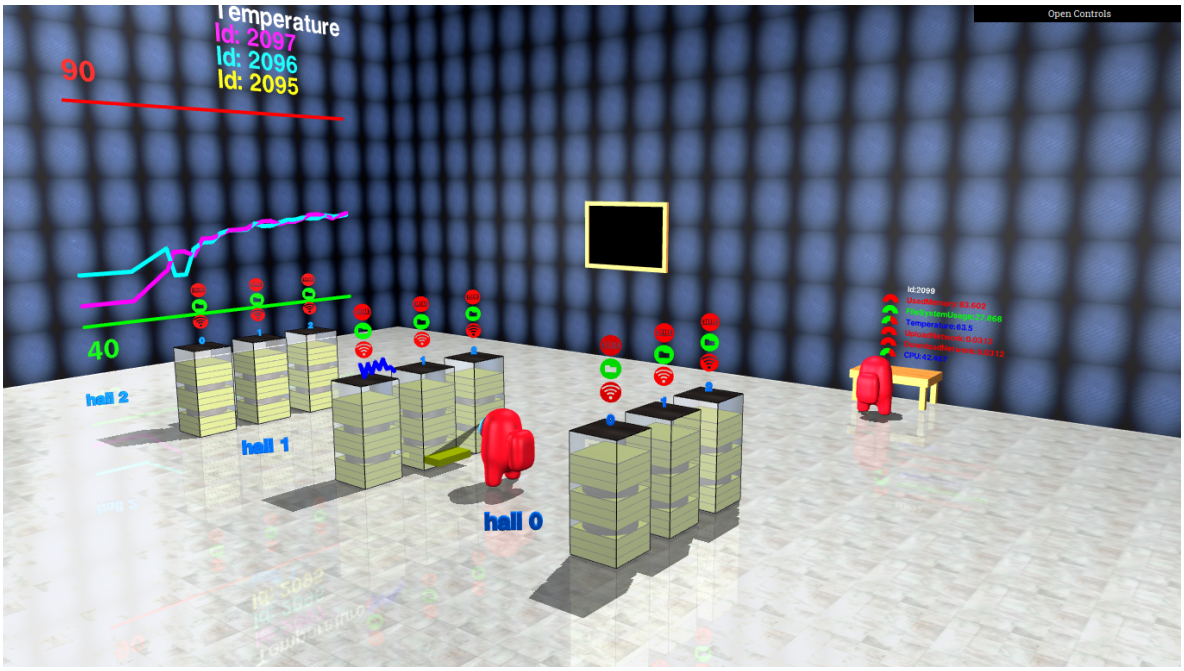

Figure 27: Room scenario view.

an EdgesGeometry to outline the object. This geometry helps distinguish its edges if the colors are similar and difficult to differentiate.

The other part of displaying an object is its material. In this case, the material is based on its color. Three values define the color of a material: red, green, and blue (RGB). In this scenario, only one Metric influences the machine's color, unlike, for example, Sunbird that every Metric impacts the final color. Six values settle the final color: the maximum and minimum limits of the exposed Metric, the machine's current performance, and three colors that the user can change on the menu. The colors are the maximum, middle, and minimum menu colors. The algorithm of the color calculation is as follows. If the machine's current performance is above the maximum limit, the color is the menu maximum color. If the machine's current performance is below the minimum limit, the color is the menu minimum color. If the interpolation is not active and the value is between the limits, the color is the menu middle color. Otherwise, if the interpolation is activated, the program mixes the maximum and minimum of the Metric, the machine's current performance, and the maximum and minimum menu colors to calculate the final color. Figure 28 displays several possibilities of the machine's color, supposing the exposed Metric has a maximum limit of one hundred and a minimum limit of forty, and the maximum and minimum colors selected on the menu are red and green, respectively. If the machine's performance is below forty, the RGB color is (0, 255, 0). If the machine's performance is over one hundred, the RGB color is (255, 0, 0). Otherwise, the green and red components fluctuate according to the machine's performance.

As the number of machines and cabinets grow, so does the number of GPU calls Three.js makes. Three.js has a solution to avoid the increase of GPU calls. InstancedMesh is a special version of Mesh with instanced rendering support. Instancing is a strategy that draws multiple objects with the same material and geometry on one render call. The technique reduces the communication time between the CPU and GPU for each object in

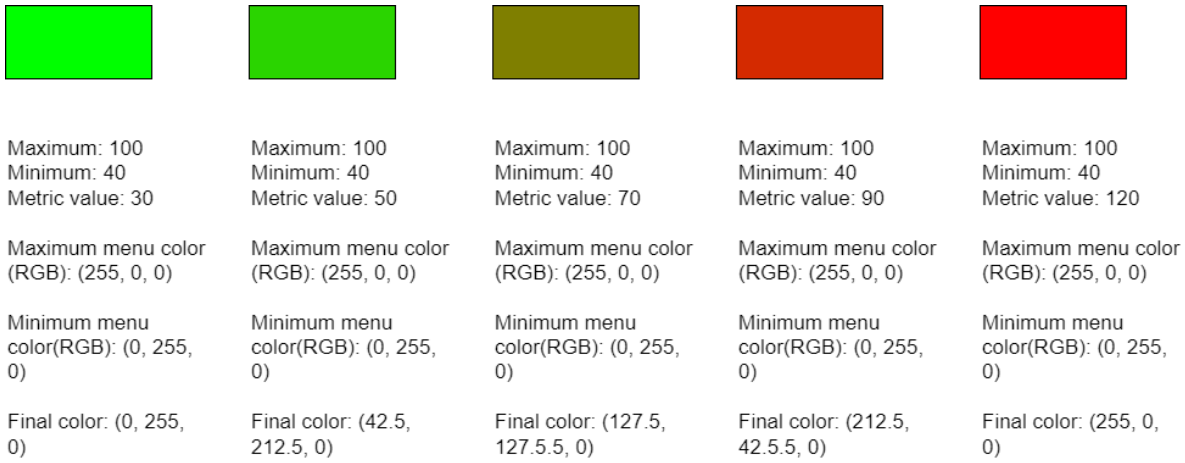

Figure 28: Example of different rack color possibilities when interpolation is activated.

each frame. For example, if the room has 200 machines, using the BoxGeometry, the plugin makes 200 GPU calls. By using the InstancedMesh, only one call is needed.

When the Room scene page is loaded, the server sends the machines' performance. As one of the scenario goals is to observe the machines' current performance, it does not make sense to refresh the page to visualize the most recent data. So, periodically the plugin requests the server to update the data.

As opposed to the machine, the cabinet's height depends on the number of machines inside the cabinet, and the color is always white and slightly opaque. Figure 29 shows a close-up image of the machines and cabinets.

4.2.2 *Inside scenario*

One characteristic of the scenario is the structure that surrounds the room itself. With Three.js to define the structure, the planes to assemble a cube need to be created. Each plane must be visible to the backside. Figure 22 illustrates those planes with textures. By default, the texture's size fits the size of the face. However, if the length and width of the room are too large, the image becomes unformatted. Nonetheless, Three.js allows repeating the texture in length and width. Therefore, to have surfaces with appropriate textures, the number of repetitions of the image is proportional to the size of the room. In the same Figure, the room reflection on the floor is visualized. By adding a mirror below the ground and making the floor slightly transparent, the floor reflects the scene.

4.2.3 *Lights*

To add shadows and help identify objects in the scene, the scenario has four-point lights positioned at the center of the walls. However, only one of them creates shadows. Otherwise, the scenario would become too confusing. A point light is a point in space that sends light in all directions. Usually, the intensity diminishes with distance

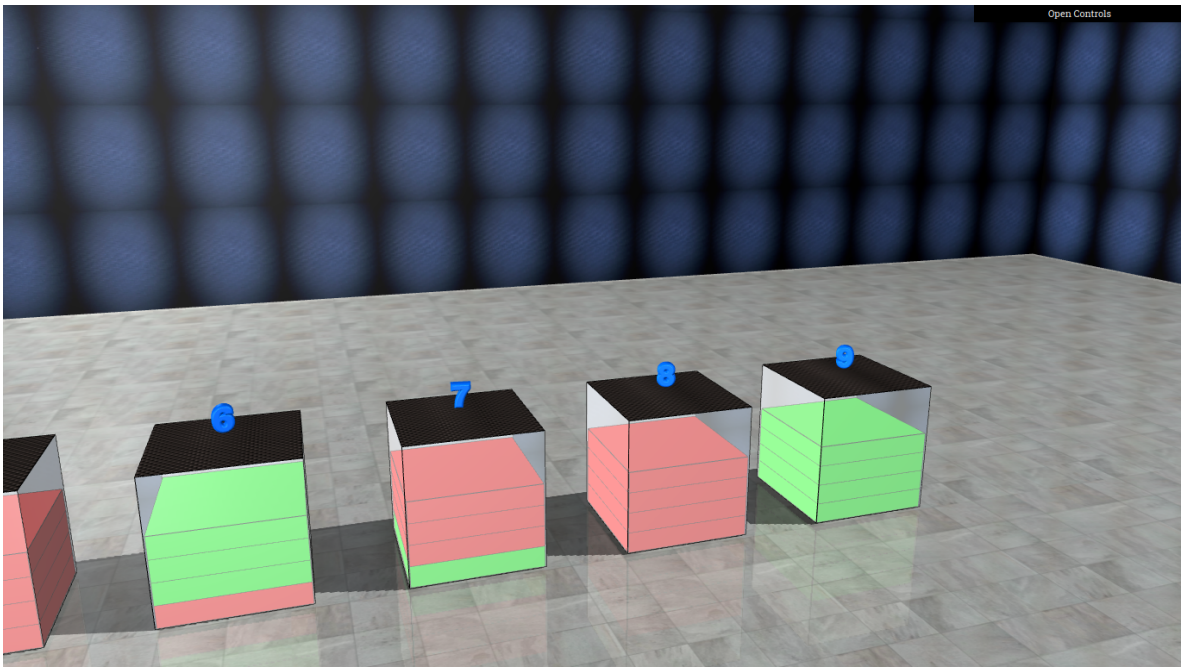

Figure 29: Machines and cabinets of the Room scenario.

from the light source, reaching zero after some space. However, in this scenario, the lights do not lose intensity across space.

4.2.4 *3D Letters signs*

Machine's geography defines their position in the scenario. However, with hundreds of machines, finding a specific machine is hard. Therefore, the scenario displays the hall's name and the cabinet indexes for easier recognition. Regarding the hall's 3D letters designation, these objects have the same x position as the cabinets that belong to the hall. The other 3D letters, the cabinets indexes, are placed on the top of the cabinet. The pre-defined Three.js text function can expose any font. With the font and a string as an input, Three.js builds a 3D object of the text. Figure 30 shows a close-up image of the 3D letters on the scenario.

4.2.5 *Warning signs*

So that the user can act as a manager of a data center, warnings of the machines with values outside the limits are displayed. The chosen procedure to notify the user is the appearance of 3D objects on top of the cabinet, as shown in Figure 31. These objects are personalized according to the Metrics of the system. The objects are an extrusion of a 2D surface. When a Metric is above or below the limit thresholds, the warning sign of the Metric is displayed. Regarding the object color, this can be one of three colors. If the concerning machines have values above the maximum limit, the color is the maximum menu color. If the concerning machines have values

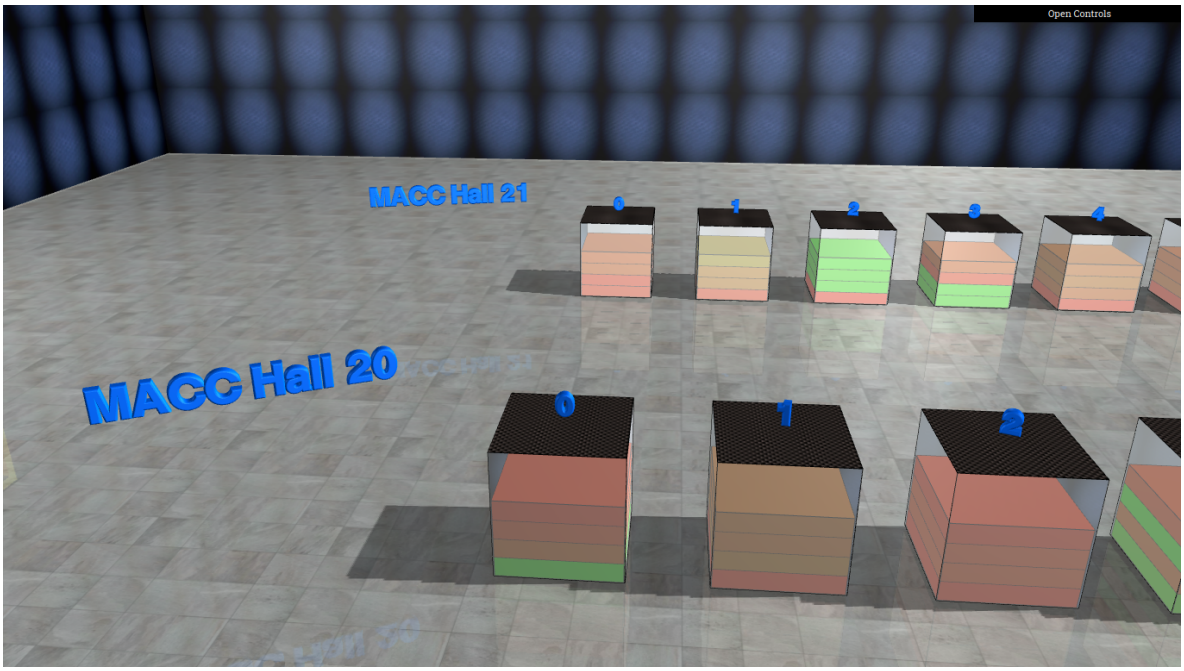

Figure 30: Close-up image of the 3D letters.

below the minimum limit, the color is the minimum menu color. Otherwise, if both cases occur, the object color is the middle menu color. Besides, the warning signs, hall names, and cabinet indexes are always oriented to the camera position.

4.2.6 *Mouse events*

A Mouse event occurs when the mouse interacts with the HTML document. Three.js has mouse event listeners to use on the scenarios. In this plugin, the mouse press and the mouse hover events are used. The algorithm for finding the machine the mouse is pointing at is the same in both situations. Three.js shoots a ray from the mouse's position and checks which machines intersect it. The first machine that intersects the ray is the one with the information displayed on the popup. If the user hovers over a machine, the program displays a popup with the machine's geography and Metrics out of bounds. Figure 32 exhibits an example of a popup with the Metrics outside of the limits and the machine's information.

If the event is mouse press, the plugin has two cases, the left and right-click. If the event is the left-click, the plugin shows a popup with the Metrics and performance information, the current task assigned, and the machine's components. The color of the performance letters depends on the menu color definitions. For example, if the performance is above the limits, the color is the maximum menu color. Figure 33 exhibits an example of a popup with the task, components, and the machine's current performance.

Besides, this popup has a button to open a new tab. Figure 34 exhibits an example of this page. The new window shows the machine's performance over time. The left side of the page has a few basic information about the machine, namely the geographic position. On the right side, multiple graphics with the machine's

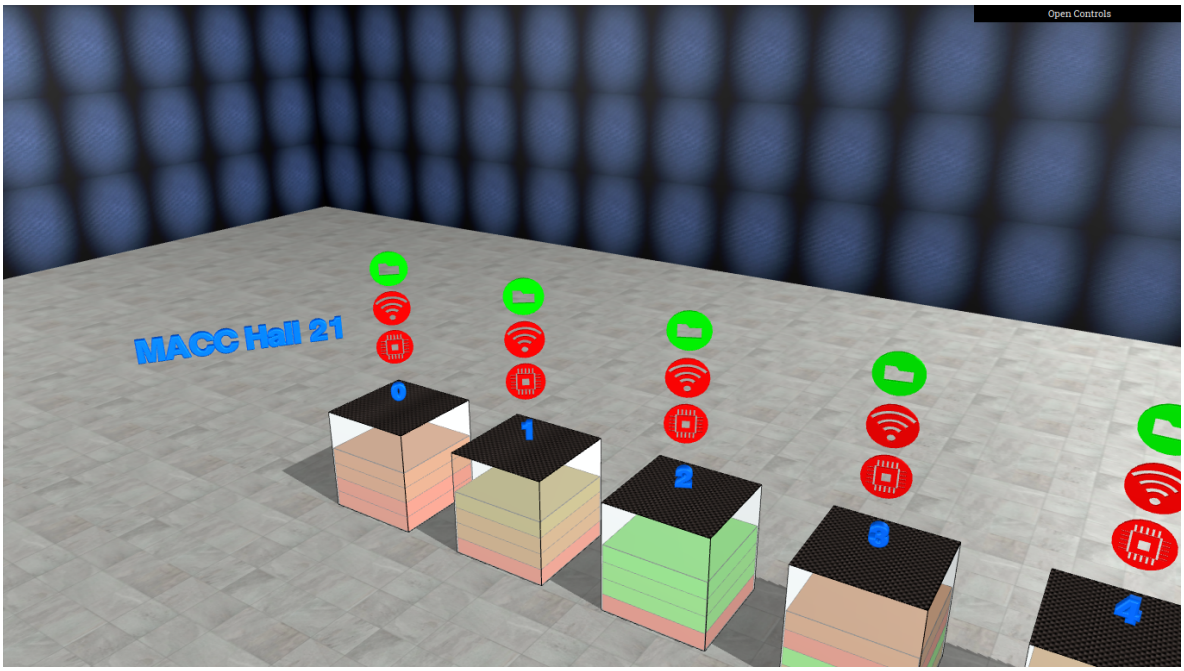

Figure 31: Warning signs appearing on top of the cabinets.

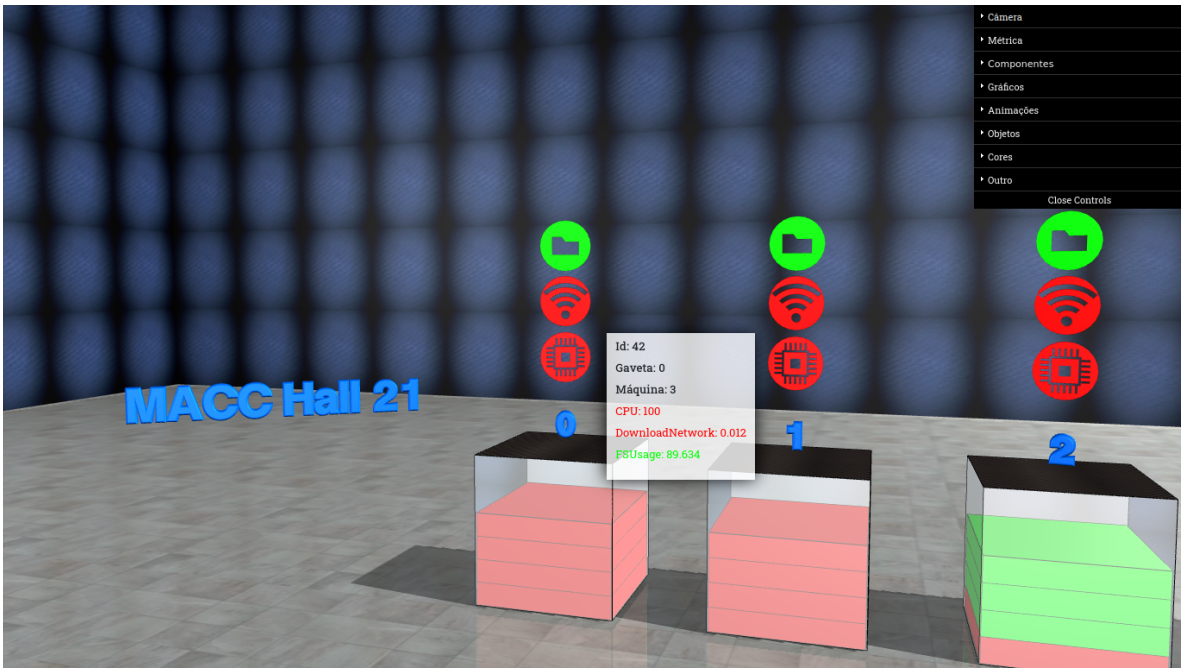

Figure 32: The popup that appears when the mouse is hovering over a machine.

Figure 33: The popup that appears when the user presses a machine.

performance over time of one Metric are exhibited. Below the graphic, a table is displayed with the Metric information, namely the name, description, unity, average, maximum, and minimum value. The rest of the page shows the performance over time of the other Metrics.

If the event is right-clicking, a simple object appears that points out the machines that have values outside the limits in the same cabinet as the intersected machine, as shown in Figure 35.

4.2.7 *The Avatar*

Since the target of this application is the regular user is crucial to make it dynamic. So the scene has an avatar that interacts with the scenario to add flow to the Room. To give the avatar movement, the object travels around the machines of the data center. The avatar's path is defined before the initial movement. The movement is composed of a list of points the avatar travels to.

The diagram of Figure 36 represents the general states of the avatar, which are the starting movement, the rotation to leave and to analyze the cabinet, walk, and analyzing the machines. After receiving the first data update, the avatar starts his movement. First, the program iterates the machines of the first cabinet and verifies if any of them have performance values outside of the limits. If the cabinet has machines out of the limits, the avatar analyzes the cabinet. Otherwise, the avatar rotates and moves to the next cabinet. When a cabinet has machines with the performance out of limits, the algorithm is as follows. First, the rack is pushed away half of its length from the cabinet. While the plugin pulls the rack, the avatar moves up to the y machine's coordinate if the machine's height is superior to the avatar's. The movement of both objects is linear, i.e., each object moves at the same speed throughout the entire path.

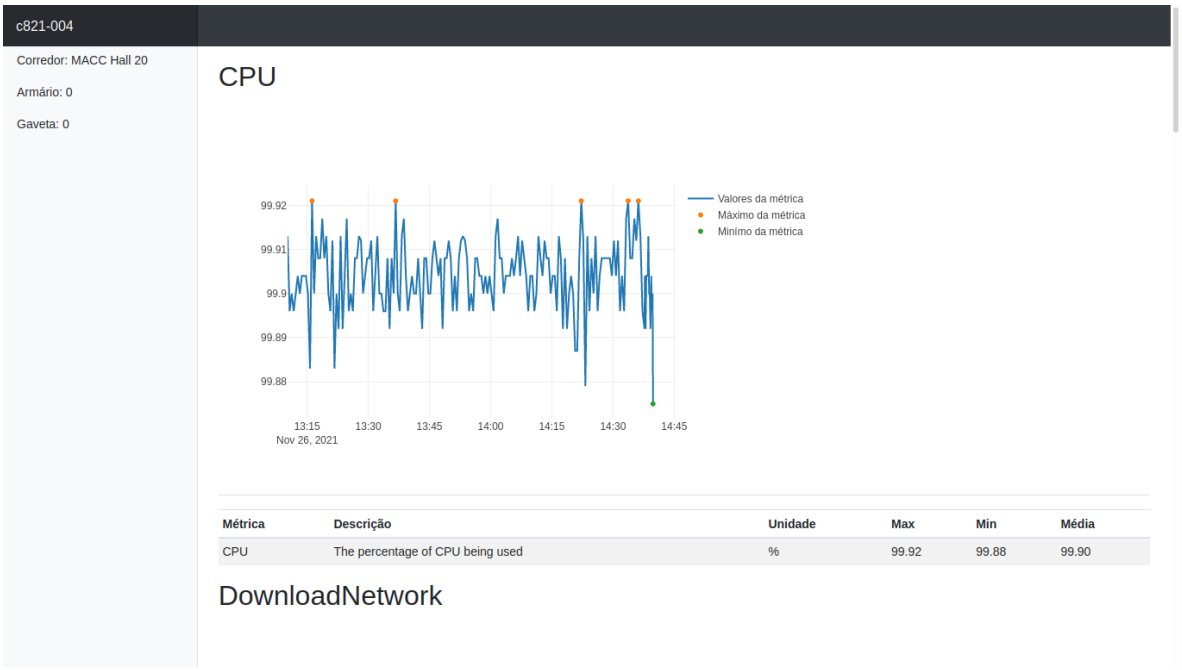

Figure 34: Page with the machine's performance over time.

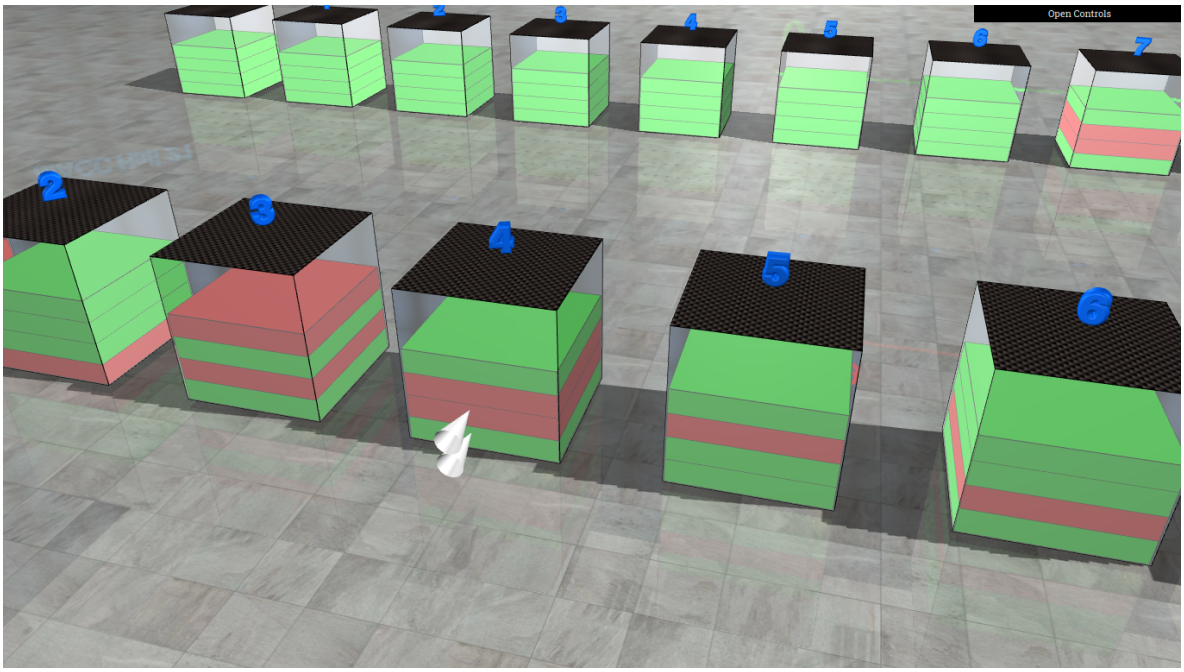

Figure 35: Objects pointing to machines with performance out of bounds.

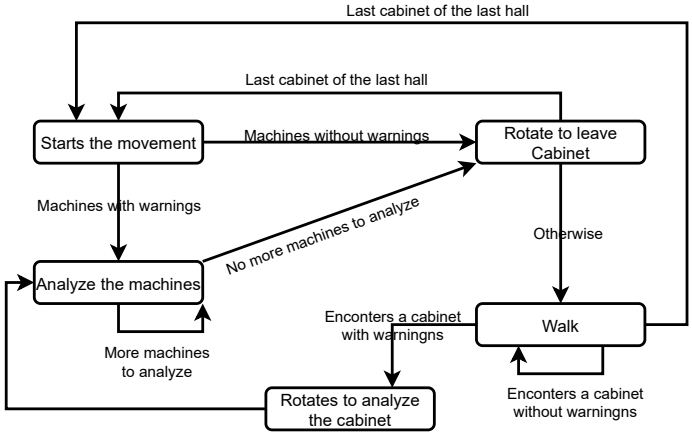

Figure 36: Avatar state machine.

After both objects finish their animation, the plugin shows the avatar scanning the rack. The scan is a simple triangle that has two vertices moving. One of the triangle vertexes positions ahead of the avatar's head, while the others are placed at the top of the analyzed rack. The vertex placed right in front of the avatar's head has equal coordinates throughout the whole process. The other two vertexes change the *x* coordinates over time to reproduce the scan animation. Figure 37 illustrates the initial position of the triangle scan.

Figure 38 illustrates the avatar analyzing a machine. The animation is divided into two parts. In the first part, the scan animation moves the two vertices of the triangle from the middle of the rack to the machine vertices closest to the avatar. In the second part of the animation, the scan makes the reverse route.

The next avatar's movement is placing the machine in its original position. This movement is similar to removing the rack but in the opposite sense. In this part, the avatar position does not change.

After analyzing all the warning machines, the avatar rotates and moves to the next cabinet. If the cabinet has no warning machines, the avatar keeps moving forward. Otherwise, the avatar rotates back to the cabinet. When the avatar reaches the end of the hall, it moves to the next hall. If the avatar reaches the last cabinet of the last hall, the avatar starts again from the first cabinet of the first hall. The rotation and walk actions depend on the index of the hall once the sense of the movement is different.

4.2.8 *Observer workstation*

In the scenario, another object like the avatar is represented, the observer. This object observes the machine's performance the avatar is evaluating. When the avatar is not analyzing any machine, nothing is presented. In this case, the observer represents the person that observes the data center, and the avatar symbolizes the Data Source. Regarding what the observer is visualizing, this is divided into two parts: a gauge graph, and the 2D letters of the Metric name and the performance associated. Figure 39 shows the exposed objects on the observer workstation.

The gauge graph indicates the percentage of the machine's performance between limits. To build the object, first, is verified if the performance is out of limits. If the performance is above the maximum limit, the torus is

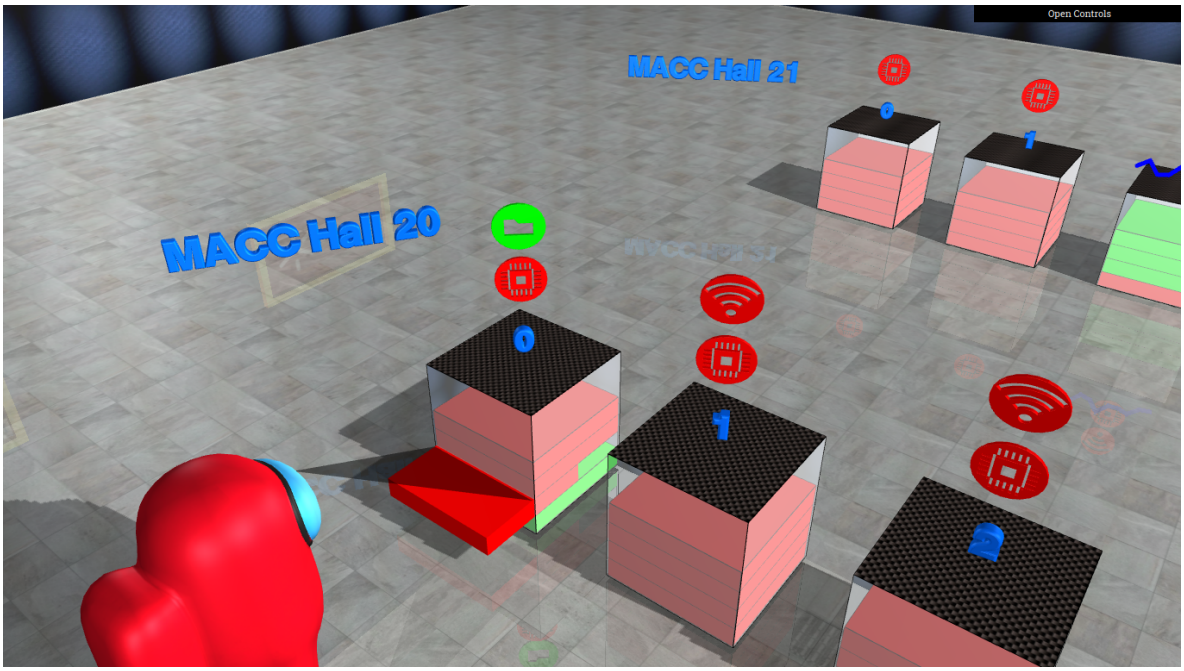

Figure 37: Avatar's initial triangle scan position.

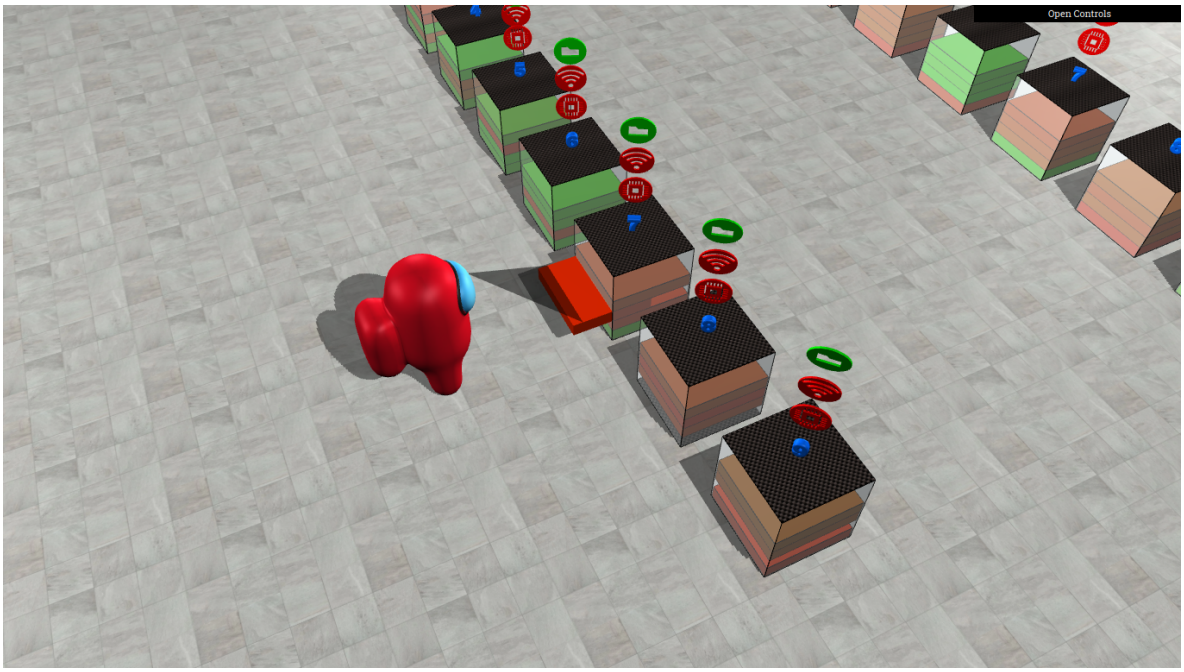

Figure 38: Avatar analyzing a machine.

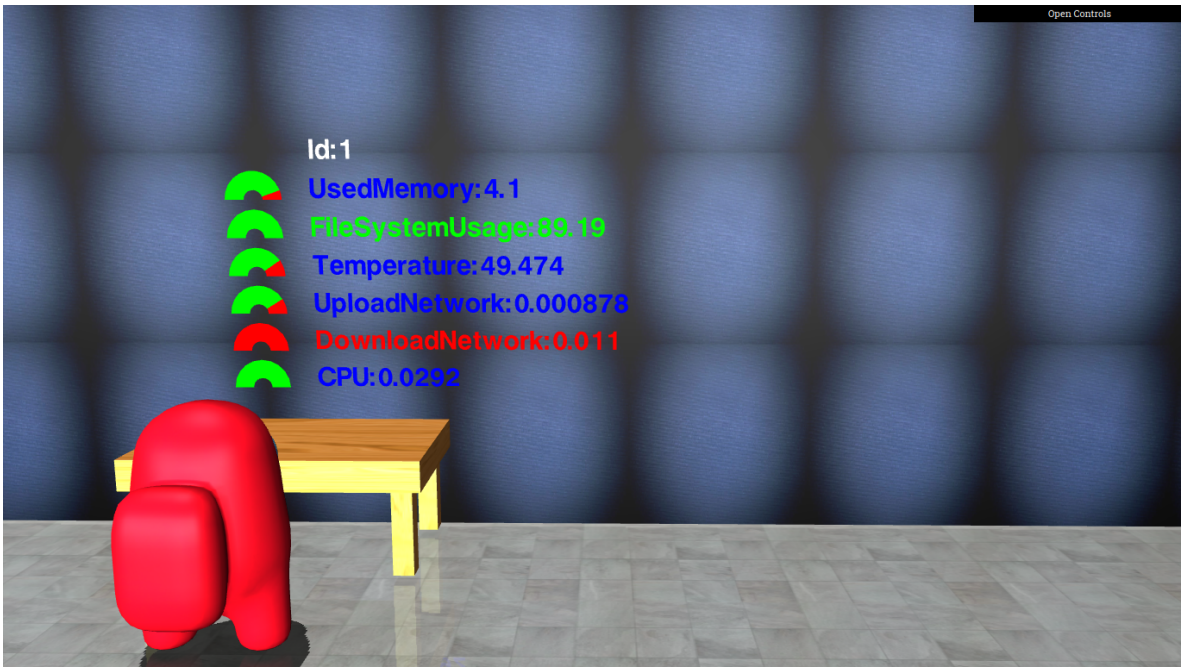

Figure 39: Observer workstation.

red. If the performance is below the minimum limit, the torus is green. Otherwise, if the value is between the limits, two partial torus are added, one with green color and another with red color. The two torus openness and rotation values are calculated on Equation 1

$$\begin{aligned} percentage\_green &= \frac{maximum - value}{maximum - minimum} \\ angle\_green &= \pi \times percentage\_green \\ angle\_red &= 1 - angle\_green \end{aligned} \tag{1}$$

where *value* is the machine's performance, and *maximum* and *minimum* are the Metric's thresholds. In the equation, the *percentage_green* ranges from 0 to 1 and indicates the green percentage of the first torus. Besides, for the red part is added another torus with *angle_red* openness and rotated in the z-axis the value of *angle_green*.

In the second part of the observer workstation, the scenario exposes the name of the Metrics and their values. In terms of aspect, the name of the Metrics is similar to the hall's name. However, these are in two dimensions. Since it takes some time to generate the Metric's name and the objects have the same aspect every time the avatar examines a machine, those are precomputed at the beginning of the scene and the visibility is changed when the avatar starts/stops analyzing a machine. Unlike the Metric names, the program generates the performance letters whenever the objects are exhibited.

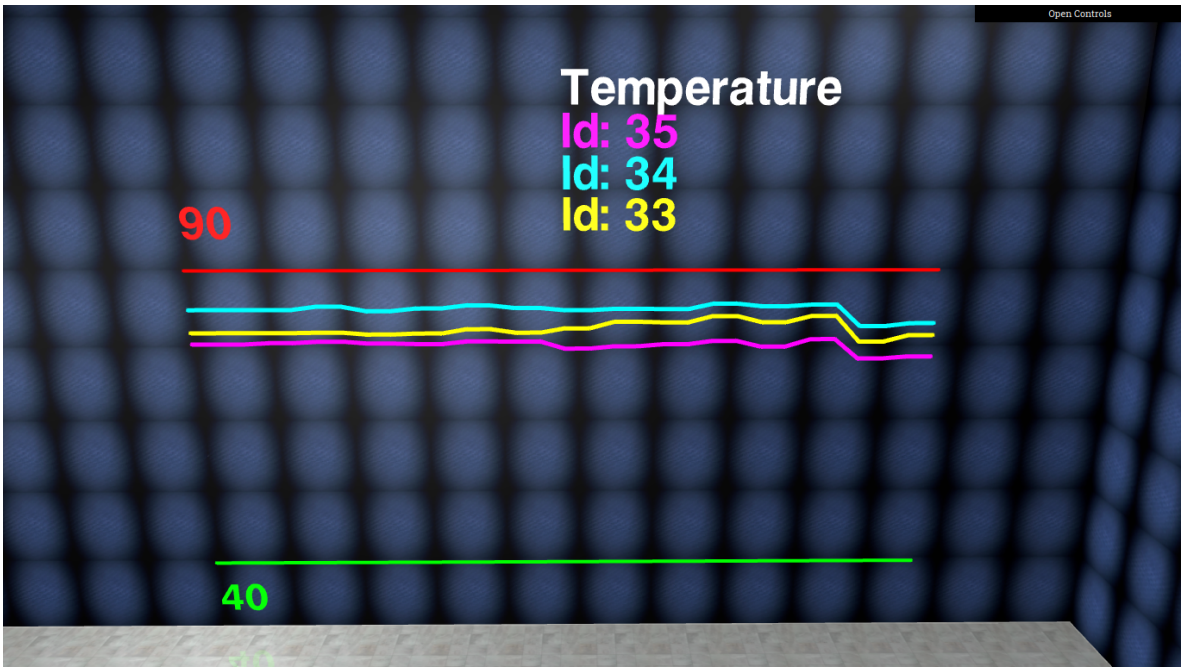

Figure 40: The graphic on the wall exhibiting the Temperature performance over time of the machines 33, 34, and 35.

4.2.9 *Graphic on the wall*

The machines' current performance is displayed on the scene through the object's color. To show the machines' performance over time, a three-dimensional line graph is created on one of the walls. Unlike the graphics in the new tabs, the graphic on the wall only shows values since the scenario was open. To identify the machines that are in the graphic, the machine's ids are displayed. The Metric's maximum and minimum values and the name are exhibited as well. The machines displayed on the wall belong to the same drawer. The graphic visualization has three alternatives: dynamic, static, and invisible. With the dynamic option, the machines displayed on the wall are changed periodically. The static option only shows the values of the drawer and Metric chose on the menu. In the invisible option, the plugin removes the graphic. The values of the graphic do not pop up instantly. During an interval, the plugin adds points to the line to build the graphic.

The lines are created with Three.js's LineMaterial. LineMaterial is a material for drawing wireframe-style geometries. Figure 40 illustrates the line graph of the Temperature performance over a period of the machines 33, 34, and 35.

4.2.10 *Tutorial*

The scenario has some functionalities that might not be easy to understand at first. So the plugin has a tutorial that explains most of the actions available in the scenario. Figure 41 illustrates the warning signs tutorial step. In each step of the tutorial, four settings are defined. The settings are the camera position, the camera look at

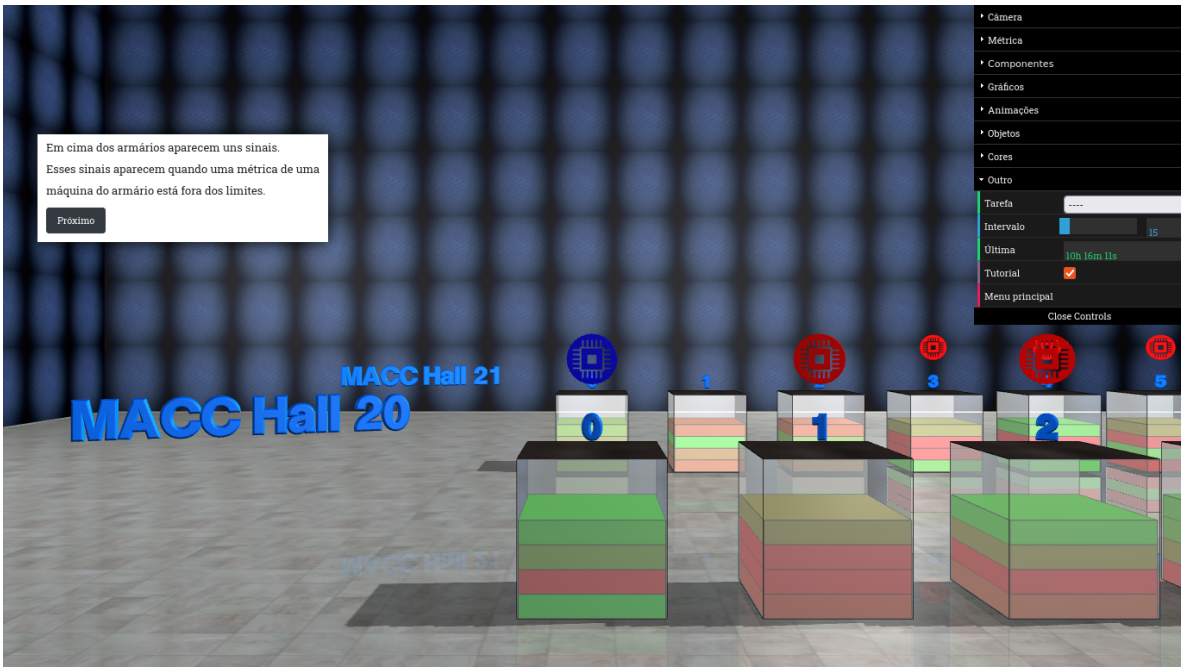

Figure 41: Warning signs tutorial step.

it, a message, and a set of actions. In each step, a set of actions are completed to animate the tutorial. The actions can be to open a menu, call functions, or change the state of the scenario. Besides, the camera is positioned at the specified place and looking at positions. At the end of each step, a message appears on a popup. Between steps, the camera position and look at position change from the location defined on the current step to the positions of the next step.

Regarding the content itself, first, the program explains what a data center is, and then the plugin features. The features detailed on the tutorial are the rack colors, limits, warnings signs, menu, the graphic on the wall, the avatar, the observer, the popups, Metrics, components, and tasks.

4.2.11 *Menu*

To build the menus of the plugin, the DAT.GUI library is used. This library is a commonly used option when building Three.js applications. In addition to menu buttons DAT.GUI allows compartmentalizing buttons by category to organize the menu. To save space Three.js can hide the menus. This plugin has the camera, metric, components, graphics, animations, objects, colors, and the other folders. Figure 42 exhibits a diagram with the plugin's folders and buttons.

First, the *camera* folder has the check button to enable the automatic camera, the type of automatic camera movement, and the default camera state button. The first button activates the automatic movement of the camera from the option selected in the second button. Three types of automatic camera movement are available on the button. On the first option, the camera looks at the center of the scene and move around the division in an elliptical movement. On the second option, the camera travels all the cabinets of the Room while looking at them.

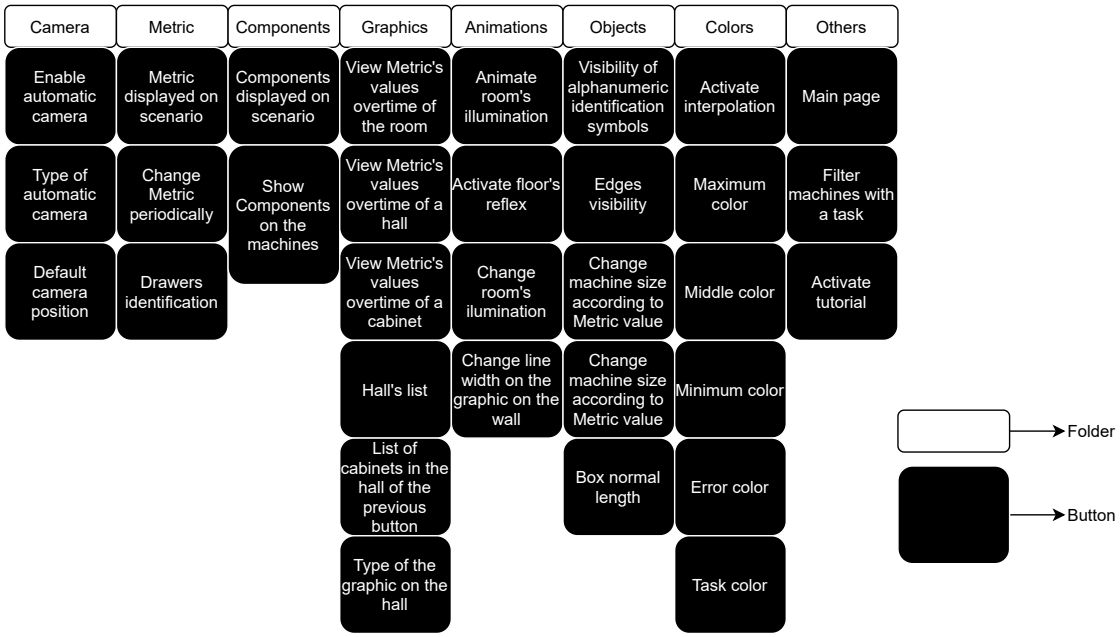

Figure 42: Diagram with the menu folders and buttons.

The third automatic camera views what the avatar is observing, i.e., the camera is positioned on the avatar's eyes. The last button returns the camera to its original position.

Next, the *metric* folder has a drop button to select the Metric to observe on the datacenter, a check button that changes the observable Metric periodically, and a drop button with all the drawers' names. When the selected Metric changes, all the information exposed in the scene changes to the new selected Metric. The drawer's name is defined by the hall name, cabinet index, and drawer index. The drawer's names are used on the graphic on the wall when the type is static.

Next, the *components* folder has a drop button with the list of components of the data center and an option to see the specification of the components exposed on the color of the machine. When the machines' color exposes the values of the specifications, the color depends on the best and worst specifications in the room. The best specifications have the maximum color, while the worst specifications have the minimum color. The other specifications have the color interpolated. If the machine does not have the component, the machine has the error color defined in the colors menu folder. If the machine has more than one specification for the component, the exposed specification is defined randomly. The last case happens, for example, when a machine has a specification of 4GB and another with 8GB of RAM.

Moreover, the *graphics* folder has three buttons to visualize in a new tab the performance over time of one Metric of a set of machines. The first button shows the machines' performance over time in the room. The second button exhibits the machines' performance over time of a hall. The last button displays the machines' performance over time of a cabinet. The page is similar to the Figure 34. The table shows the machine's information exposed in the graphic above the table. Two more drop buttons on the folder select a hall and a

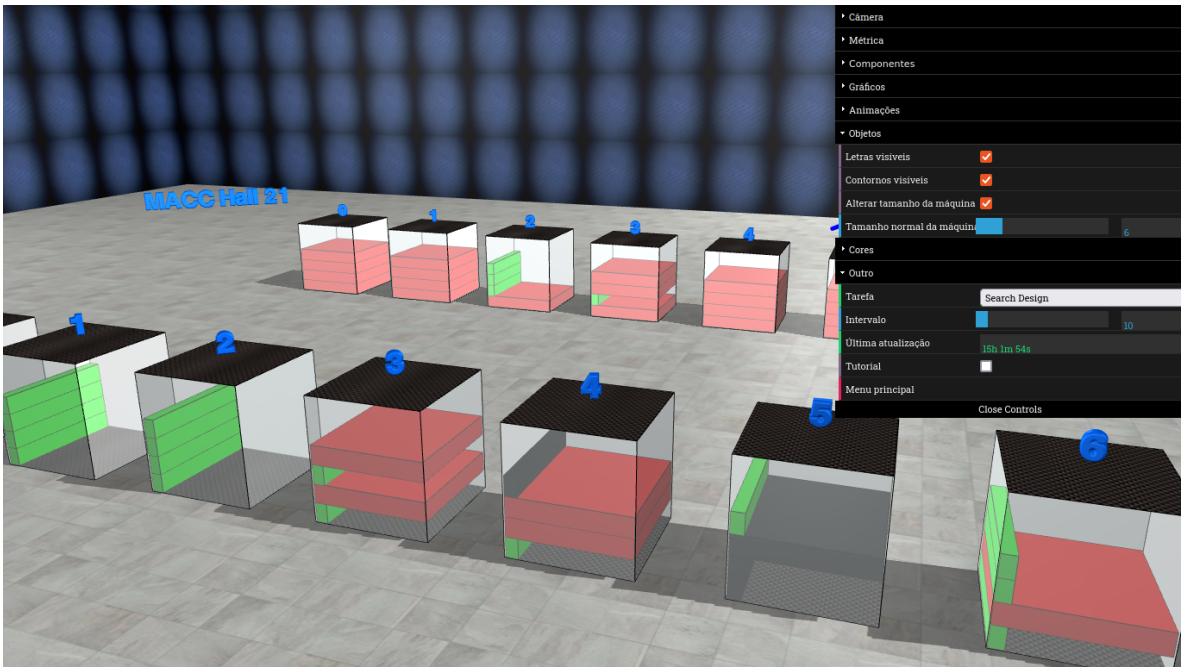

Figure 43: Scenario view with change machine size button activated and a task selected on the menu.

cabinet to expose the Metrics on the new tab. Lastly, it has a drop button that selects the type of graphic on the wall, the dynamic, static, or invisible.

Following, the *animations* folder has a few options to make the scenario a little different. The change lights button animates the illumination's color according to the global state of the room. The activate reflex button hides or shows the reflex of the room on the floor. If the user considers that the scenario does not have an accurate illumination, it can change the intensity of the lights. And, the last option is to change the line width of the lines on the graphic on the wall.

Subsequently, the *objects* folder has a few buttons to change the state of the objects in the room. The identification words button puts the visibility of the alphanumeric symbols according to the button state. The edges visible button puts the visibility of the machines and cabinets edges according to the button state. The change machine size button, when checked, changes the machine size according to the percentage of the machine's current performance between the thresholds. If the button is unchecked, the rack scale is normal. Otherwise, the size changes according to the maximum and minimum values of the Metric and the machine's performance.

For example, if a machine is thirty percent of the maximum limit, then the machine size is thirty percent of the default size. If the percentage is above one hundred, the size is the standard. If the percentage is below ten percent, the machine only has ten percent of the usual size. Figure 43 shows the machines with different sizes. The last button of the folder is the default box size. This button changes the typical machine and cabinet size to one on a range of values.

Next, the *colors* folder has a button that enables/disables the interpolation option. With this option deactivated, the colors of the machines between the limits are the middle color, as previously mentioned. The following five

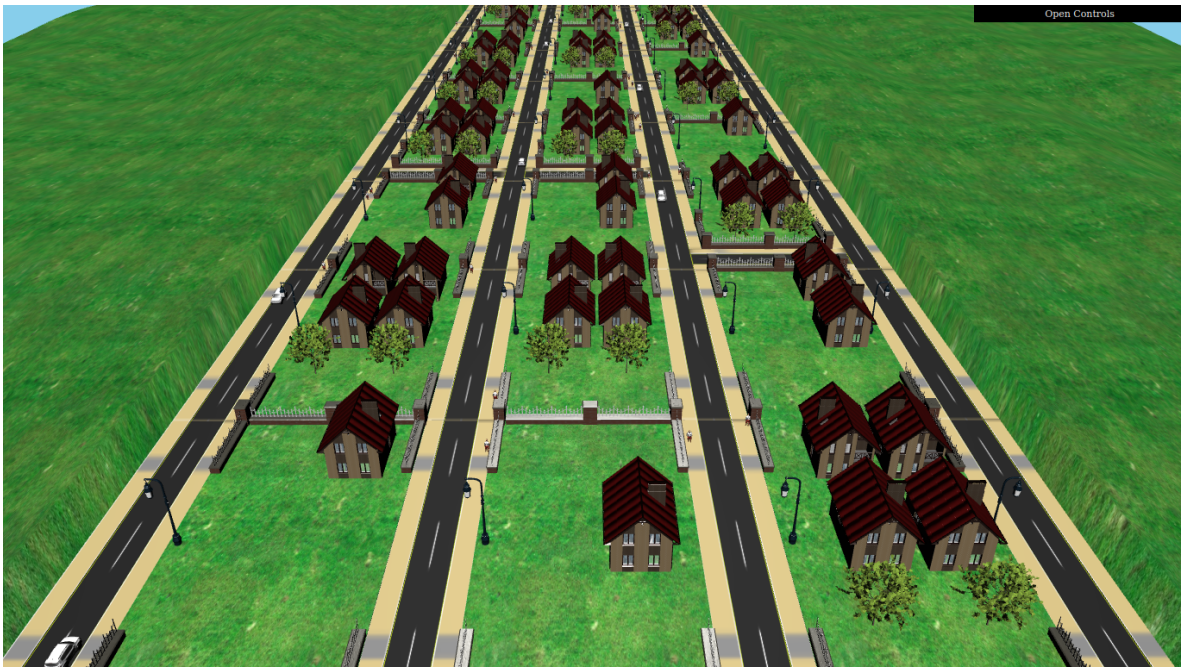

Figure 44: City scenario view example.

buttons of the folder are color picker buttons: the minimum, the maximum, the middle, the error, and the task colors.

The last folder is the *others* folder. This folder has a button to choose the interval at which the data is updated, when the last update occurred, a button to come back to the main page, filter the machines with a specific task, and the tutorial check button to activate/deactivate it. Figure 43 illustrates a set of machines with black color once they have the task selected in the menu.

4.3 CITY

This plugin represents the physical data center in an unusual scenario to show that the application can apply the data center information to any scene. The chosen way to represent the data center is as a city.

Figure 44 and Figure 45 illustrate an example of the City scenario based on one room of the data center. The Figure illustrates houses, persons, cars, walls, roads, lamps, and trees. The houses represent the machines of the data center. The persons and cars represent the data flow between the machines and the outside world. The more data the machine exchanges over the network, the more cars and persons move to and from that house. Moreover, the horizontal roads separate different halls of the room. Beyond that, the walls have the goal to separate different drawers. The other objects, namely the lamps and, trees are on the scene for aesthetic reasons.

Figure 45: Cars and people representation on the scenario.

4.3.1 *Object positioning*

Figure 46 illustrates the city structure. Each horizontal line on Figure 46 represents a hall. In each hall, a piece is added for every drawer, i.e., the red rectangles. The pieces are surrounded by roads to enable the cars' movement within the lines. The first drawer of the cabinet has a road at the left. The drawers in the same cabinet do not have roads separating them but walls, represented as white lines on the diagram. For example, in Figure 46 the line with index zero has three cabinets, hence it has three vertical roads. The first and the second cabinet have three drawers, and the last cabinet has one drawer. In each piece, four machines of the same drawer are displayed. If the drawer has more than four machines, the drawer occupies more than one piece, which is the case of the drawer (1,3) and (1, 4). Those pieces do not have a white line separating them, which means they belong to the same drawer.

The first operation fulfilled in the scenario is the 3D model loading. The objects represented by 3D models are houses, trees, walls, roads, lamps, men, and cars. When the program loads every model to the client, the plugin maps the roads and positions every house, road, lamp, tree, and wall.

The field surrounding the City has its height based on a grayscale image. The brighter the value of a pixel on the image, the taller the correspondent point on the field. The image is produced with Perlin noise to create a realistic plane Perlin (1985).

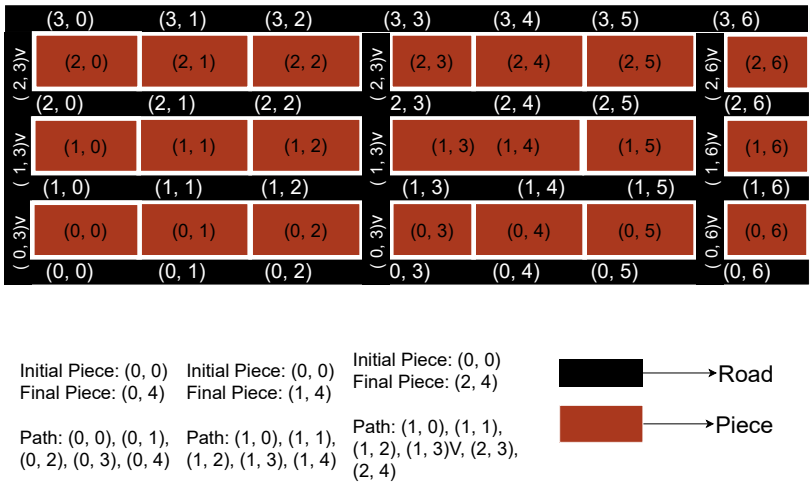

Figure 46: Diagram with the road indexes and pieces. The first number is the hall index, and the second is the index in the hall.

4.3.2 *Path definition for cars/people*

As mentioned before, the scenario has cars and people that move around the city to represent the data flow on the data center. The cars and persons move between houses or a house and the outside world.

First, the plugin creates objects to move on the scene and defines the initial and final pieces according to the machine upload and download. The more downloads a piece has, the more objects will have it as its destination. The more upload a piece has, the more objects will start on the piece. With the initial and final pieces defined, the plugin decides if the object is a car or a person and if the initial or final positions change to the exterior world.

Then, the plugin iterates every object to define the pieces of the movement. Using the structure in Figure 46 as a basis, several examples of possible paths are defined. An object that starts on piece (0, 0) and finishes on (0, 4) will have as path the roads (0, 0), (0, 1), (0, 2), (0, 3) and (0, 4). An object that starts (0, 0) and finishes on (1, 4) will have as path the roads (1, 0), (1, 1), (1, 2), (1, 3) and (1, 4), because the piece (0, 0) has access to the road (1, 0). An object that starts (0, 0) and finishes on (2, 4) will have as path the roads (1, 0), (1, 1), (1, 2), (1, 3)V, (2, 3) and (2, 4).

Based on those examples, the algorithm in Figure 47 is developed. When the object starts the algorithm, it checks if one of the surrounding roads has direct access to the final piece. Direct access means the object only uses a horizontal road to reach the destination. If the object has direct access, the algorithm adds the segments that allow the object to reach the final piece. If the object does not have direct access, the algorithm chooses the surrounding segment closer to the final piece. Then the algorithm adds the segments that provide a path for the object to reach a vertical segment. The next step is to add the vertical segments that allow the object to have direct access. Finally, add the horizontal segments to reach the destination. Each piece on the object's path has a direction associated (up, down, left, or right), and which side of the road or sidewalk the object moves to help his movement within the piece.

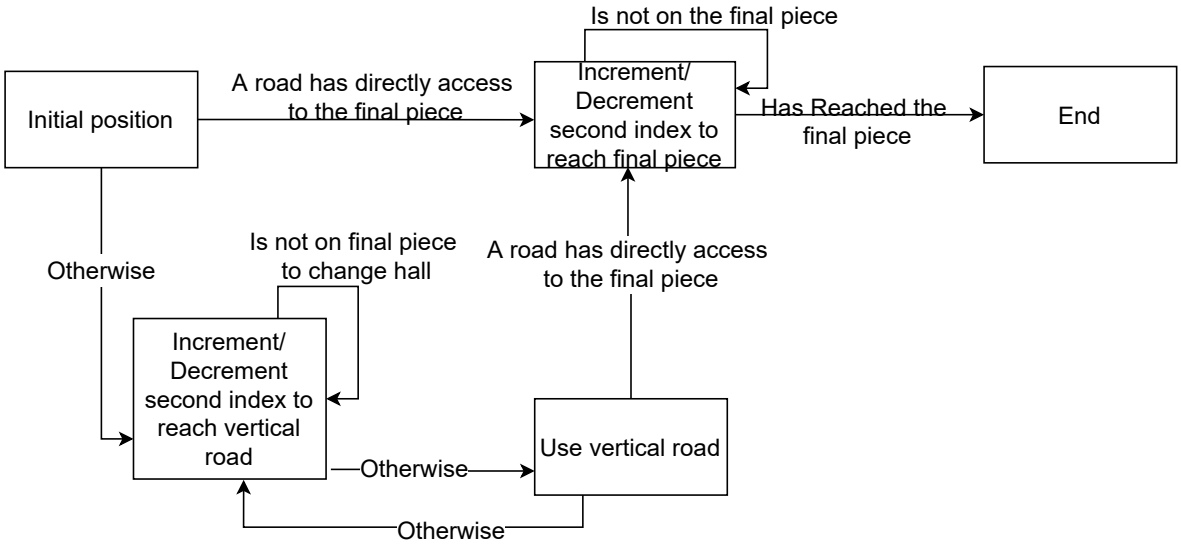

Figure 47: Algorithm to define the object's path pieces.

4.3.3 *Motion algorithm*

After defining which pieces are on the object's movement, the plugin checks if the objects can be added to the scene, i.e., if they have collisions with any object that is already on the roads. If no object prevents its movement, then the house the object left changes the color to red to signalize an object is leaving the house. The object is also added to a map that contains the pieces the objects are located. Figure 48 shows a house that has an object that will leave it (red color). If an object prevents its addition to the scene, it will be checked again later. Periodically, new cars and persons are added to the scene.

After an object enters the scene, it moves within the roads. The movement of objects in the scene is divided into three parts: the first movement inside the piece, moving within the piece and leaving the piece.

First, when the object makes its first movement within the piece, the current time is saved, and the time that it takes to run through the piece is calculated.

Second, the object moves to a new position within the piece, according to the initial time, the current time, and the time it takes to run through the piece. Next, the collisions between the object and other objects close to its location are verified. The analysis depends on the type and directions of the objects. Collisions occur in two different situations. In the first case, the collision may occur with objects moving in perpendicular directions. In the second situation, the collision can occur with objects of the same type and direction, as the object ahead may be immobile. An object stops when another object prevents its movement. For example, a person is crossing the road, and a car is waiting until it crosses. If the car has a car behind it, both need to wait.

Third, when an object is leaving a piece, the plugin updates the object's piece on the map to the new piece. If the object goes to the last piece of its movement, the house that the object arrives in changes to green to signal the house is receiving an object. Figure 48 illustrate a house that will receive an object. If the piece is the last piece of the object's movement, the object is removed from the scene.

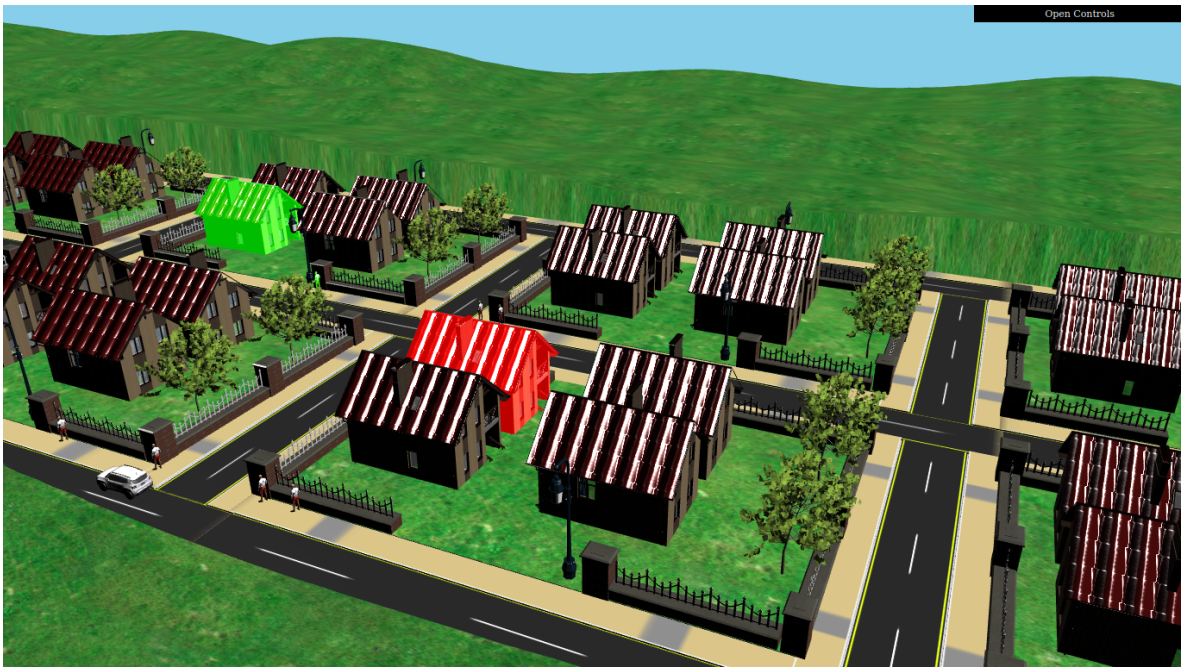

Figure 48: Houses changing their colors to show a piece is leaving (red) or arriving (green).

Despite the algorithm working well in cases of little congestion, it has a problem when the roads have a lot of congestion. The City scenario has a deadlock case when objects that depend on others create a cycle. For example, the object with id 35 depends on object 45, the last depends on the 55, and this on 35. The objects enter on deadlock state and stay stuck forever. To solve the problem, when an object can't move, the object id points to the object id that prevents the movement. In the end, the application iterates the ids of the immobile objects and the list they depend on. If in that iteration the object id appears more than once, it means the object is on a deadlock. The program removes from the scene the objects that causes the deadlock. In the previous example, the program removes the object with id 35 and solves the deadlock issue.

4.4 SUMMARY

This chapter detailed the work made on the front end of the app. The chapter started by describing how to create a Visualization plugin to illustrate the data center information. The application has two plugins, as an example, to display the data center data. In the Room scenario, the plugin illustrates the data center machines in a setting closer to reality. In each machine, the scenario shows their assignment, the components, and the current performance values. In new tabs, the plugin displays the machines' performance over time. The City scenario exposes the data center devices as houses in a city. The house's position follows the machine's geography on the data center. Between the houses and the exterior world, the machines exchange objects to represent the data flow within the data center. This scenario shows that from the data center information any scene can be created.

5

BACK-END IMPLEMENTATION

This chapter explains the tools used on the application, namely the database and the Data Source in Section 5.1. The models the application uses to save the data center structure and machine's performance values are described in Section 5.2. Later, Section 5.3 the routes the server handles requests are exposed. Then, the employed algorithms to get the performance from the Data Source are illustrated in Section 5.4 and Section 5.5. Next, in Section 5.6, how to create a Data Source Connection Plugin is described. After that, how to run the application is illustrated in Section 5.7. Finally, the unit tests realized on the back-end to validate the algorithms are described in Section 5.8.

5.1 TOOLS

Figure 49 shows the implemented application architecture with the Prometheus server, the PostgreSQL replication, the Prometheus Connection Plugin, and the Visualization Plugins, Room, and City.

Prometheus is used to keep track of the state of machines by observing the devices and their performance. It has several advantages that explain its use in large companies and why it is standard for monitoring. Prometheus has a multi-dimensional data model with time series data identified by Metric name and key/value pairs, making it fast to observe the machine's performance in a period. Besides, Prometheus has no reliance on distributed storage.

The data is obtained from Prometheus with an HTTP request with a query in PromQL, and the response comes in JSON format. If the request is about the machine's current performance, the response has an address list with the current value and time. If the request is about the machine's performance over a period, the response is a list of addresses made up of a set of times and values.

The relational database management system (RDBMS) used is PostgreSQL. As the database handles many queries from the servers, the database is replicated, i.e., the application uses more than one DBMS instance. Database replication increases availability, scalability, performance and avoids a single point of failure. The replication architecture selected in this system is the master-worker architecture Thomas (2020).

Figure 50 illustrates the database system architecture, which has three main layers: PostgreSQL, Patroni, and HAProxy. First, the RDBMS layer, PostgreSQL, stores the data. Second, Patroni is a software layer that reduces the reliance on a single point of failure, i.e., acts as a master coordinator. Patroni also uses *etcd* to store the location of the primary server. If the primary fails, *etcd* handles the election of the new primary server and

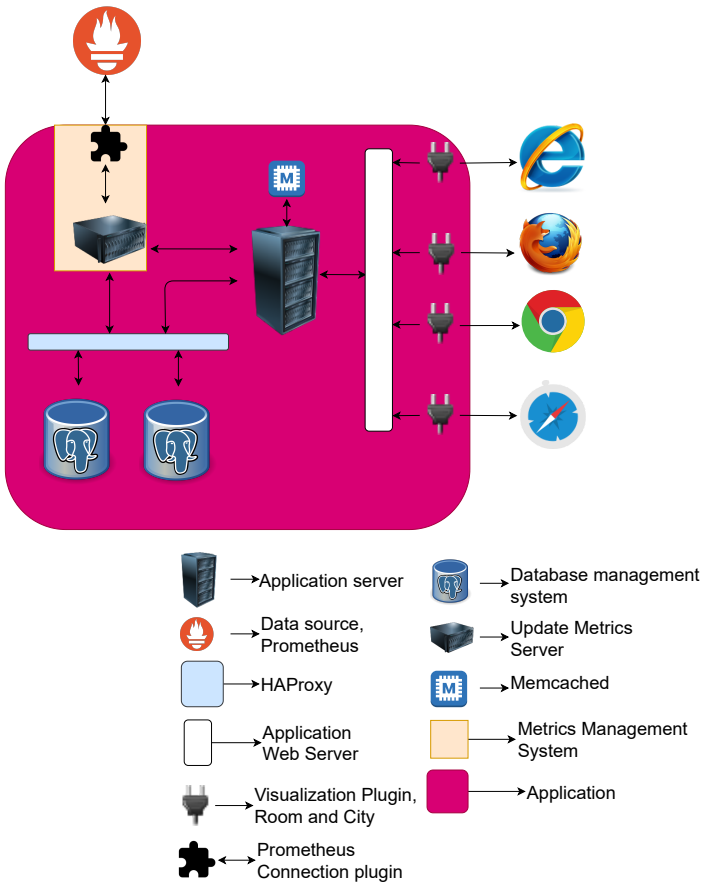

Figure 49: Implemented application architecture.

guarantees that only one becomes the new master. Finally, the HAProxy is a routing layer that ensures the write queries always reach the primary node. The HAProxy also spreads read queries across multiple PostgreSQL replicas. Thomas (2020) also has a recipe to backup the data to prevent information loss in case of catastrophic failure.

5.2 DATA MODELS

The application stores the inventory and the machine's performance on the database to be used by the Application servers and the Update Metrics Server. In Flask, models correspond to the tables on the database. In Flask creating the models on the database automatically is possible.

5.2.1 *Inventory Database*

The Inventory Database saves the information about the data center structure. Figure 51 demonstrates the conceptual model of the Inventory Database, which is composed of a Room, Hall, Cabinet, Drawer, Rack, Task,

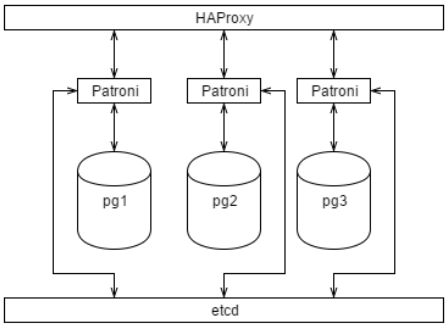

Figure 50: Database architecture. Source Thomas (2020).

Specification, and Component. A Room is composed of a set of Halls. A Hall has multiple Cabinets, which are composed of Drawers. A Drawer has a list of Racks. The Room, Hall, Cabinet, Drawer, and Rack indexes define the rack's geographic position within the data center. Those entities have an id and a designation. Beyond that, the Rack entity has an address that is the machine's identifier on the Data Source.

Additionally, the Task contains information about the current projects assigned to the data center. First, the Task designation identifies the project name. Next, the description characterizes the assignment in detail. Finally, the link is a reference to a page with more information about the Task. Each machine has only one project, but a project can have multiple machines operating the Task.

Ultimately, the Component represents the hardware of the devices, like RAM, CPU. The Specification is the different versions of the Component. For example, for the RAM component, are available specifications like *RAM G.SKILL Aegis 8GB (1x8GB) DDR4-3000MHz CL16*. A Rack can have several Specifications, and a Specification can be in several machines. The conceptual model is converted automatically into a logic model (Figure 52) that defines the models of the system.

5.2.2 *Performance Values Database*

The Performance Values Database stores the information about the machine's performance. Figure 53 demonstrates the conceptual model of the Performance Values Database, which is composed of a RoomValue, RackValue, Date, and Metric.

The conceptual model of this database has some similarities with the Inventory Database, namely the RoomValue and RackValue entities. The application exports the rooms and racks on the other database into this one. The RoomValue is defined by an id from the Inventory Database, and the attribute *last_instant_request_time*, that identifies when occurred the last Metric update of the room. The RackValue entity has the id, address, and *last_range_request_time* attributes. The id and address have the same function as the other database. The *last_range_request_time* is used to avoid making unnecessary requests to the Data Source about the performance values over time. This time means that the database possesses all the machine's performance data older than this time. The geographic location does not matter here (apart from the racks in the Room), so no entity is represented between the Room and the Rack entities.

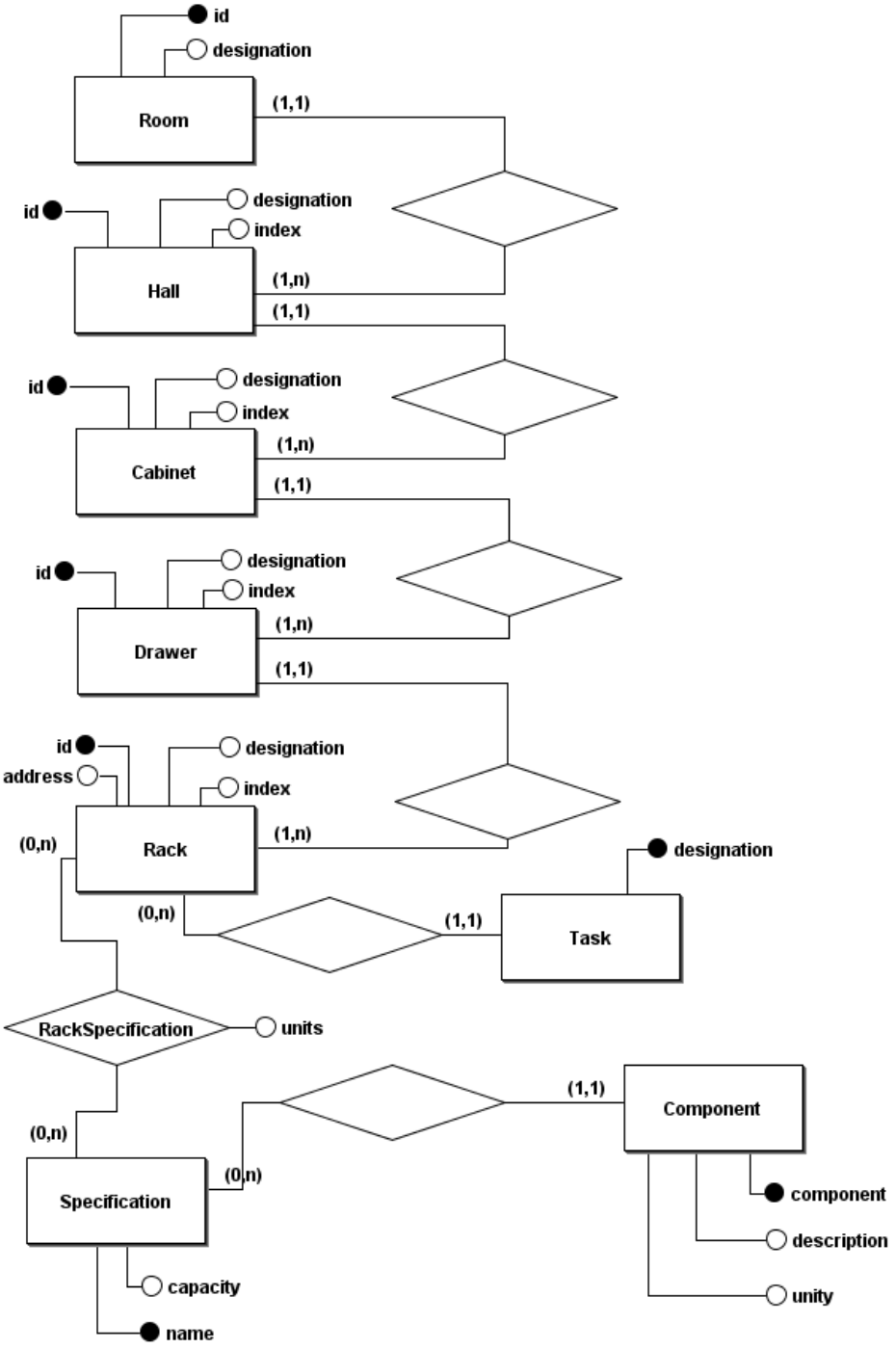

Figure 51: Inventory Database conceptual model.

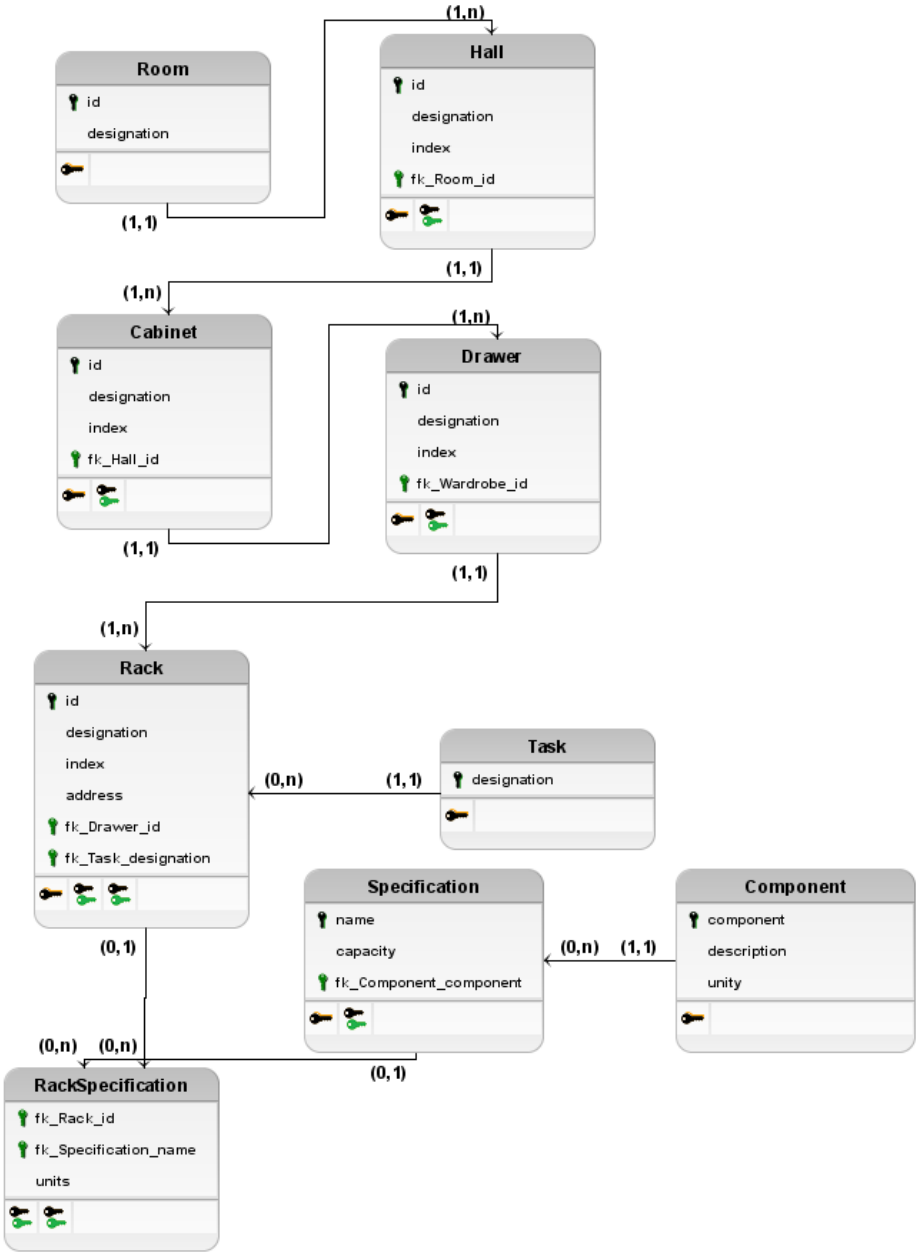

Figure 52: Inventory Database logical model.

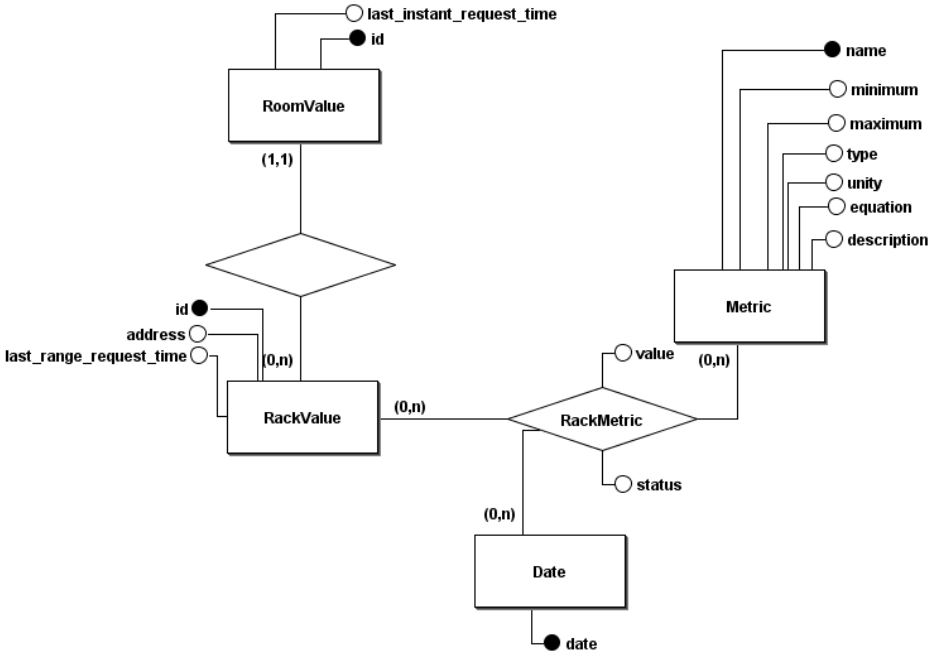

Figure 53: Performance Values Database conceptual model.

Furthermore, the Metric entity represents the measurement that evaluates the machine's condition or resource usage. The database has the Metric entity with the name, minimum, maximum, type, unit, equation, and description attributes. First, the name is the Metric identification. Next, the *minimum* and *maximum* attributes are the thresholds in which the machine performance is concerned. Following the *unity* is the unity of measurement of the Metric. Succeeding the *description* is the Metric definition to help the user understand what the Metric is measuring and how it can be valuable. Subsequent, the *type* key is the group that the Metric belongs to. For example, the download and upload Metrics belong to the network category, Finally, the *equation* is the Data Source query. The entity Date has a date as an attribute to define the time. The three entities, Date, Metric, and RackValue, create a relation called RackMetric that defines the status and a value from a machine (RackValue) in a specific time (Date) of a particular Metric. The conceptual model is converted automatically into a logic model (Figure 54) that defines the models of the system.

5.3 ROUTES

In Flask, routes are the handlers that deal with the client's requests. The application has a set of routes that define the operations the clients can make. The routes respond to the client either with an HTML file or a JSON message. The HTML files display the scenarios while the JSON message sends the information about the data center's structure, machine's performance, Tasks, Components, or Metrics. The following items exhibit the APIs defined for the Application Server.

- **main menu** Displays the app's main page;

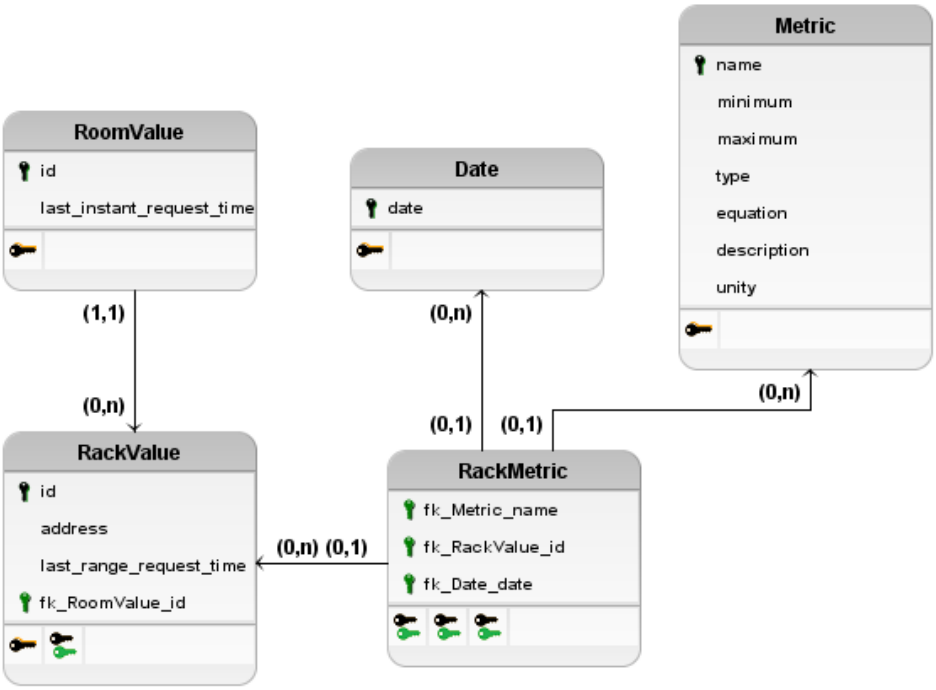

Figure 54: Performance Values Database logical model.

- **scenario** This route renders any HTML file. The first segment of the URL is the handler identifier. The second segment is the name of the rendered template. The rest of the segments can be used as information to the scene. For example, for the */scenario/Metrics/1*, the room with id 1 is rendered using the *Metrics.html* file;
- **rooms info** This route sends a JSON with a list of the rooms ids and designations;
- **room structure** This route receives a room id on the headers. The server sends the client a JSON dictionary with the room structure;
- **specific info** This route receives a list of names. The server sends to the client the extra files info that is on the list of names;
- **tasks** This route sends all the tasks data in JSON format;
- **components** This route sends the list of components and their specifications in JSON format;
- **metrics** This route sends the list of Metrics in JSON format;
- **instant room** This route receives an identifier of the room. Sends the current performance of the machines in the room in JSON;
- **instant machines metrics** This route receives a list of machines id's and Metrics names and sends the current performance of those machines and Metrics. The information is sent in JSON;

- **range machine** This route sends in JSON a machine's performance over time;
- **range machines metrics** This route takes a list of machine ids and Metric names and sends the performance over time of those machines and Metrics. The information is sent in JSON;

Furthermore, it has some routes to handle server errors, e.g., page not found, to hide the problems from the client.

The algorithm is similar in every handler. The handlers are the instant room, instant machines metrics, range machine, and range machines metrics. The route receives on the request the machines and metrics. The server checks if the input is valid. For example, the client could send a machine that does not exist. If the input is valid, the server retrieves the data from the Memcached server. If Memcached does not have the data, the Application Server retrieves the data from the database. In this case, the data is inserted on the Memcached to be used later. Finally, the information is sent to the client in JSON format.

5.4 UPDATING THE MACHINES' CURRENT PERFORMANCE VALUES

The data about the machine's current performance need to be updated regularly. So, periodically the Update Metrics Server asks Prometheus about the latest performance values of the machines and inserts them on the database.

The algorithm to get the machines' current performance is as follows. Every *x* seconds, the Update Metrics Server wakes up and checks the rooms the clients visited at a specific time. If the clients did not visit a room, the server discards that room in this request. If the rooms discarded has a request before the next iteration, it will be on the list of rooms to demand the current performance to the Data Source. Then, the program gets all of the addresses from the machines of the rooms to ask. Subsequently, the Update Metrics Server will make a request per Metric. The Update Metrics Server calls the Prometheus Connection Plugin to request the Data Source. Then, the Update Metrics Server checks for all machines from the Data Source response. If the machine does not have a value on it, the Data Source does not possess the information about the machine. The absence of the value indicates a machine has a problem. The machine value inserted in the database will be error. Lastly, the *last_instant_request_time* value on each requested room is updated to the current time. The values are added to the database and committed at once to guarantee that all machines have the latest performance values, or none have it.

5.5 OBTAINING THE PERFORMANCE VALUES OVER A PERIOD

When a client asks the Application Server for performance values over time for a machine, it checks the database for periods without data. If the database has no data about the machine in certain periods, the Application Server sends a message to the Update Metrics Server to retrieve it from the Data Source. When the Update Metrics Server receives the message, it puts the machine and their Metrics in a queue to request the Data Source later.

After updating the machines' instant performance, the application requests the Data Source about the performance values over time. The algorithm applied depends on the queue size. If the queue is below a specific threshold, the Update Metrics Server requests the Data Source multiple times for each machine for the missing information. If the queue length is above a specific threshold, the Update Metrics Server requests the Data Source once. This request asks for an extended period to include every piece of missing data of all machines. For example, considering the threshold is ten minutes, and the queue has two machines. One machine has missing data for the last hour. The other machine has missing data for the penultimate hour. The algorithm chooses the first option and makes two requests. On the first request, the server asks the Data Source about the data of the first machine on the last hour. On the second request, the server asks the Data Source about the data of the second machine on the penultimate hour. In another example, the queue has ten different machines. Each machine has missing data spreading for the last three hours. The algorithm selects the second option and makes one request about the last three hours.

The applied algorithm depends on the queue size due to the amount of data exchanged and the number of requests. If the queue is below a specific threshold, the Update Metrics Server requests each rack individually in specific periods to reduce the data exchanged between Data Source and the Update Metrics Server. If the number is above the threshold, making multiple requests per rack slows the process down due to a large number of requests. The Update Metric server avoids making too many requests by asking for all the data of the racks at once. The downside is the amount of repeated information that comes with it. If a rack has only a chunk of missing data, the request comes with many repeated values.

5.6 DATA SOURCE CONNECTION PLUGIN

The Data Source Connection Plugin is used by the Update Metrics Server to retrieve the machine's performance from the Data Source. The system allows different Data Sources once the Data Source Connection Plugin is replaceable. This plugin has two tasks: asking the Data Source about the machine's current performance and the machine's performance over time.

To create a Data Source Plugin, the following steps are required. First, create the class DataSource that inheritances the DataSourceInterface. Then, the plugin must have the method *request_instant_metric*. The method requests the Data Source about the current performance of multiple machines. This method takes as arguments a Metric, addresses, and the location of the Data Source. The Metric is the resource asked to the database. The addresses are the machine's addresses asked to the Data Source. The Data Source location is where to contact the Data Source.

Next, the plugin must have the method *request_range_metric* defined. The method requests the Data Source about the range performance of a set of machines. Besides the arguments of the first method, this method has start and end times. These times define the range of data obtained from the Data Source. Both methods must return a similar data structure to which they are associated, as Listing 5.1 shows.

```
class DataSource(DataSourceInterface):

    def request_instant_metric(metric, addresses, data_source_url):
        # Must return the following structure
        [{'metric': {'instance': '2.148.56.199'}, 'value': 10}, {'metric': {'
            instance': '54.154.180.224'}, 'value':  25}, ...]

    def request_range_metric(metric, addresses, start, end, data_source_url):
        # Must return the following structure
        [{'metric': {'instance': '2.148.56.199'}, 'values': [[1640338205142, 10],
             [1640338255142, 12]]}, {'metric': {'instance': '54.154.180.224'}, '
            values': [[1640338205142, 25], [1640338255142, 22]]}]
```

Listing 5.1: Data Source Connection Plugin requirements.

5.7 SERVER CONFIGURATIONS

The Python virtual environment is necessary to run the application. A virtual environment is a Python environment that isolates the dependencies of different projects. The virtual environment allows running other libraries versions on the same machine for different projects. With the virtual environment installed and activated, the device needs to download the libraries on the virtual environment. Instead of installing the Python packages manually, Python does it with just one command.

To facilitate the program maintenance and configurability, the application has a JSON file to fill in some information. It specifies the database and Data Source locations, among other information. If the JSON file is not configured, the application does not know where to ask or store the data.

Listing 5.2 exhibits an example of the configuration file. The first set of fields, *DATABASE_INVENTORY_LINK* and *DATABASE_VALUE_LINK* are the URLs to connect with the databases. The Update Metrics Server and Memcached location are specified in the second set. The expressions in the third set provide the following information: the Data Source server location, the plugin that handles the Data Source requests, the file with the Metric's configuration, and the log file, respectively.

The last set of fields contains information about the system's behavior. First, the *INTERVAL_REQUESTS* is the interval in seconds the Update Metrics Server inserts the machines' performance values on the database. In this case, every ten seconds, the Update Metrics Server requests the Data Source to update the machines' current performance. Next, the *TIME_SAVED_DATABASE* is the period in minutes that the data is available to the clients. According to the configuration of the Listing 5.2, the machines' performance of the last ninety minutes is available to the client. Moreover, the room needs to have a request in the last *TIME_TO_DROP_ROOM_FROM_REQUEST* in minutes, or the machines' current performance of that room is not updated. The *EMPTY_DATA* is the interval with no data in which a machine is considered to have missing data. At last, *BEGIN_TIME* and *END_TIME* is the interval at which the data source has data about the performance of the machines. If those fields are defined, the system gets the machines' performance on that interval.

Metrics are measures that assess the state of a machine. For the administrator to configure the Metrics that appear in the application, these are characterized in a YAML file. The file *CONFIG_FILE* specified on Listing 5.2

```
"DATABASE_INVENTORY_LINK": "postgresql://user:pwd@ip:port/datacenter",
"DATABASE_VALUES_LINK": "postgresql://user:pwd@ip:port/datacenter_values",

"SECONDARY_HOST":"127.0.1.1",
"SECONDARY_PORT":"1236",
"MEMCACHE": "127.0.0.1",
"MEMCACHE_PORT": "11211",

"DATASOURCE_URL":"127.0.0.1:9090",
"DATA_SOURCE": "prometheus",
"CONFIG_FILE":"configuration/macc_config.yaml",
"LOGFILE":"logfile.log",

"INTERVAL_REQUESTS":"10",
"TIME_SAVED_DATABASE":"90",
"TIME_TO_DROP_ROOM_FROM_REQUEST":"300",
"EMPTY_DATA":"600",
"BEGIN_TIME": "2020-05-26 19:00:00",
"END_TIME": "2020-06-05 09:00:00",
```

Listing 5.2: Example of a configuration file on *config.json*.

is particularized for that. The file must have the same structure as Listing 5.3, which is an example of the Metrics configuration.

So that the administrator can send other types of data to the front-end, it can define a set of information in files. In this case, the predefined information specified is the form of the warning signs exposed in the front-end of the application. According to Metric *type*, a custom signal to warn the user is defined. The shapes must be specified on the *CONFIG_FILE*, as is present on Listing 5.3 on *extra_info* key. The *extra_info* key has a list of key values. The pair represents the type name and the file that contains the points of the figure, respectively. The *type* on the Metrics definition addressed previously must correspond to one of the keys on the list. For example, on Listing 5.3 the DownloadNetwork Metric has the Network type present on the extra info list. Despite the extra info being employed just for the warning signs, it can be used with other types of information.

The file in Listing 5.4 is composed of a set of shapes. Each shape has a set of points and holes. The points define the outside mold, while the holes define the interior shape. For this example, the FileSystem, the points mark a circumference, and the holes define the image of a file.

5.8 UNIT TESTS

Unit testing is a testing method that allows putting the code under various tests to examine whether the system is ready for production. Python has a framework called *unittest* that allows the creation of multiple functions that requests the routes and any method on the server. At the end of each test, the *unittest* module compares the expected value with the output value, and if the values are different, the test or the method is incorrect. For this application, the *unittest* allowed, for example, the discovery of an error on the request range performance method on the Update Metrics Server. The project has ninety-five tests, and all of them passed. Besides validating the

```
metrics:
- FSUsage:
    minimum: 5
    maximum: 75
    equation: "100 * (avg by (instance) (node_filesystem_free_bytes{device
        =\"/dev/sda5\",__filter__} / node_filesystem_size_bytes{device=\"/dev/
        sda5\",__filter__}))"
    unity: "%"
    description: "The percentage of CPU being used"
    type: "filesystem"

- DownloadNetwork:
    minimum: 5
    maximum: 50
    unity: "Mb"
    description: "The instant value of download in megabits"
    type: "network"
    equation: "sum by (instance) ((irate(node_network_receive_bytes_total{
        __filter__}[__scrap-interval__])) / (1024 * 1024))"

extra_info:
  network: "app/static/shapes/network.yaml"
  filesystem: "app/static/shapes/filesystem.yaml"
```

Listing 5.3: Example of Metric definition and the files that contains the extra information.

application's algorithms, the unit tests also check if the system keeps working when the database, Data Source, Memcached, and Update Metrics Server are unavailable.

5.9 SUMMARY

This chapter described the tools used on flexible elements, namely the database with PostgreSQL and the Data Source with Prometheus. Next, the chapter described the Flask models that define the inventory and values databases. Later, the routes in which the server handles requests, such as the scenario, the range, and instant performance requests, and the different machine's information are exposed. Then, the employed algorithms to get the instant and range metrics from the Data Source are described. Next, how to create a Data Source Connection Plugin is illustrated. After that, how to run the servers, set the configuration file fields, install the packages, and set the Data Source is detailed. Finally, the Python unit tests realized on the back-end to validate the algorithms are described.

```
shapes:
 - shape1:
     points:
         - [110, 0]
         - [106.25184089179751, 28.470094961277283]
         - [95.26279441628826, 54.99999999999999]
         - [77.78174593052023, 77.78174593052022]
         - [55.000000000000014, 95.26279441628824]
         - [28.470094961277283, 106.25184089179751]
         - [6.735557395310443e-15, 110]
         - [-28.470094961277294, 106.25184089179751]
         - [-54.99999999999998, 95.26279441628826]
         - [-77.78174593052022, 77.78174593052023]
         - [-95.26279441628826, 54.99999999999999]
         - [-106.2518408917975, 28.47009496127731]
         - [-110, 1.3471114790620885e-14]
         - [-106.25184089179753, -28.470094961277237]
         - [-95.26279441628824, -55.000000000000014]
         - [-77.78174593052024, -77.78174593052022]
         - [-55.00000000000005, -95.26279441628822]
         - [-28.47009496127727, -106.25184089179751]
         - [-2.0206672185931328e-14, -110]
         - [28.470094961277233, -106.25184089179753]
         - [55.000000000000014, -95.26279441628824]
         - [77.7817459305202, -77.78174593052024]
         - [95.26279441628822, -55.00000000000005]
         - [106.25184089179751, -28.470094961277276]
         - [110, 0]
     holes:
         - hole1:
             - [-60, -40]
             - [-60, 40]
             - [-15, 40]
             - [0, 20]
             - [60, 20]
             - [60, -40]
             - [-60, -40]
```

Listing 5.4: File system shape definition file.

6

EVALUATION

To validate the application and its content, the application is tested with data from a real data center. MACC as Minho Advanced Computing center is a Portuguese infrastructure that supports scientific investigation in supercomputing, data science, and visualization. MACC is a public institution, so it needs to be transparent about its operation.

This chapter explains the extent to which the application is suitable for showing MACC data to the general public on Section 6.1. Next, Section 6.2 exposes the performance tests realized on the application.

6.1 USE CASE

To test the application, MACC provided the machine's performance data of their data center. The data source of the MACC information is the Prometheus server. To not reveal the IP addresses of the machines, MACC replaced the IP with a code of their geographic location. To populate the data center in the database, a separate script requests Prometheus. In the request, the machine's addresses are collected. Each machine address is fragmented into several parts to define the room, hall, cabinet, drawer, and rack positions.

As MACC did not provide the current task of each machine, the tasks are assigned randomly. Since tasks run on more than one machine and communicate with each other, they are placed consecutively, between two and six machines, due to communication efficiency. Periodically, the Update Metrics Server updates the task of the machine with new assignments. All the tasks on the application can be found on the RNCA, the Rede Nacional De Computação Avançada FCCN.

MACC did not share information about the components of their data center. So, to exhibit the components, several components and specifications to use as an example are created. The components created are the processor, RAM, GPU, and disk.

The main goal of the project is to show the data center's operations to the general people. With the MACC example, the different forms of data exposed on the application scenarios of the data center are evident. More specifically, the MACC projects, the machines' performance, and components are displayed.

6.2 PERFORMANCE TESTS

At the end of the implementation, to know the Application Server limits, performance tests were conducted. Many alternatives are available to conduct performance tests on a web server, such as JMeter, Taurus, Locust, and WebLOAD. The tool used to test the webserver is Locust. Locust is an open-source load testing tool designed in Python. The performance tests conducted are written in Python, unlike JMeter that has a GUI. Locust conducts the performance tests by creating several users that request the tested server. The code is composed of a set of functions with the requests to carry out to the tested server. Each function has a weight that defines the odds of a user calling it. The defined functions are:

- **main page** This function calls the main page of the application. The function has weight 2;
- **rooms info** This function requests the server about the rooms and their id and designation. The function has weight 2;
- **room structure** This function requests the server about the structure of a room. The function has weight 2;
- **specific info** This function requests the server about the info that is on the extra files. The function weight is 1;
- **tasks** This function requests the server about the tasks on the data center. It has weight 1;
- **components** This function requests the server about the components of the machines. The function weight is 1;
- **metrics** This function requests the server about the Metric's exposed on the application. It has weight 2;
- **instant metric room** This function requests the server about the machines' current performance of a room. The function has weight 10;
- **instant metric machines** This function requests the server about the instant performance values of a set of machines and Metrics. It has weight 4;
- **range machine** This function requests the performance values over time of a machine. The function has weight 4;
- **range metric machines** This function requests the server about the performance values over time of a set of machines and Metrics. It has weight 3;
- **room** This function requests the server about the HTML page of a room scenario. It has weight 1;
- **city** This function requests the server about the HTML page of a city scenario. It has weight 1;
- **js, image, CSS** This function requests some of the js, images and CSS files. The function has weight 1;

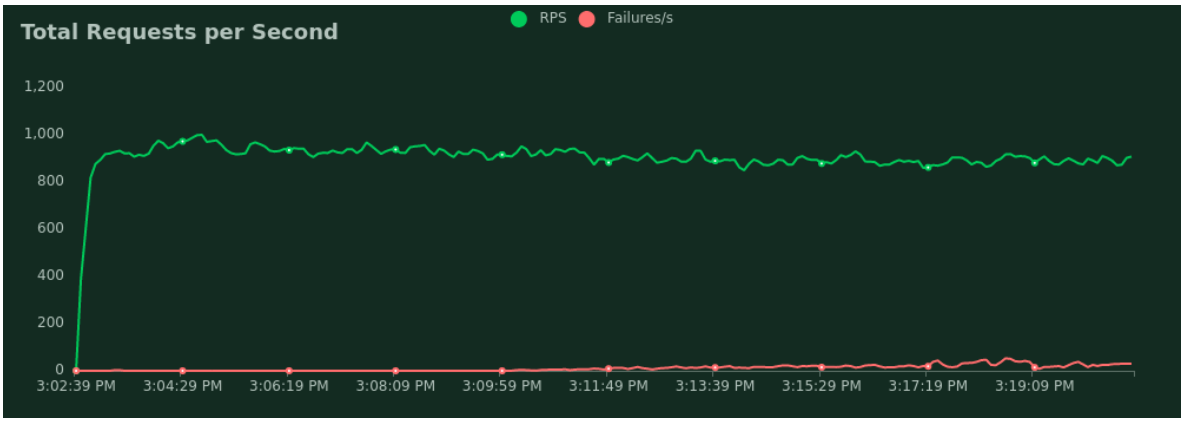

Figure 55: Total requests per second from the Locust test.

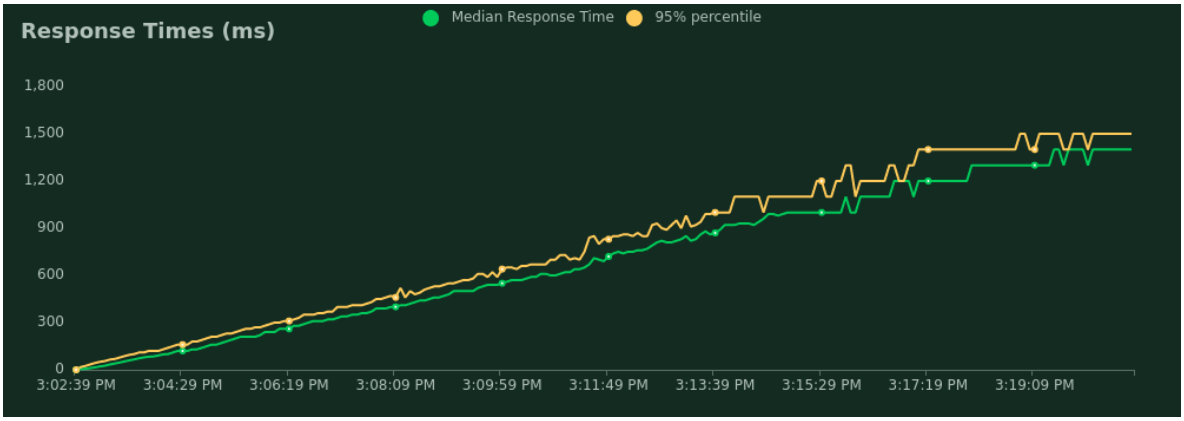

Figure 56: Response times from the Locust test.

After defining the requests, Locust gives an option to select the number of users that are asking the server and the number of users that spawn per second. To test the Application Server is defined one thousand users with spawn rate one. Figure 55, Figure 56 and Figure 57 illustrates the Locust's performance tests on the server.

The first graph shows the number of requests and the number of failures that the server handles per second. The second graphic illustrates the average response times. The last graphic exhibits the total number of users that are requesting the server. With the data from the graphs, the following points are deducted: The more users request the application concurrently, the greater the number of failures. The more users request the app, the bigger the average response times. If the goal is to have an average response time lower than a second, then the number of users requesting the Application Server can not be above eight hundred.

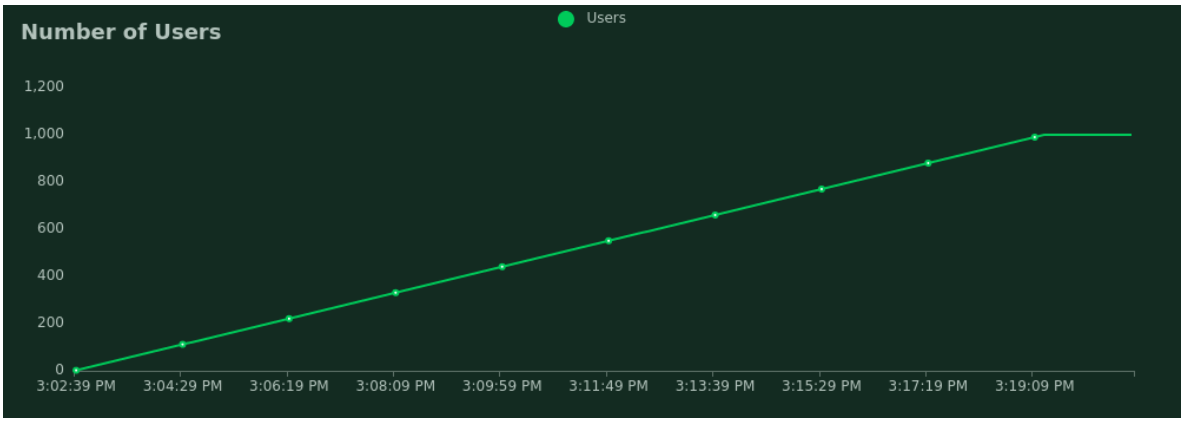

Figure 57: Number of concurrent users requesting the server from the Locust test server.

6.3 SUMMARY

This chapter described the use case of the application with a real data center. By last, the chapter illustrated the performance tests realized on the program with the locust framework. The system with a single Application Server can handle a significant workload.

7

CONCLUSIONS AND FUTURE WORK

7.1 CONCLUSIONS

Investments in data centers by governments have increased to develop new technology. Therefore allowing people in general to have a peek at the data center activity can provide a global acceptance of the funding required to run these centers. Some tools that expose information from a data center are available. However, those applications are restricted to a set of people and are not capable of receiving hundreds of concurrent requests. Also, those applications do not have layouts oriented to the general people as their goal is to be observed by the data center managers. Besides, these applications have somewhat a limited visualization capability. If the data center manager wants to observe/expose other types of visualization, those applications are not capable of displaying it.

To expose the data center activity, a web application is developed to be used by different data centers to exhibit its information to the general public. The application should expose the machines' tasks, performance, and components.

The system is composed of several components to help its isolation, maintainability, and scalability. To exhibit the machine's performance data, the data center must use a Data Source Server. As the application uses Prometheus as the default Data Source, it could be an obstacle for several data centers that do not use Prometheus as their Data Source. However, through a set of instructions, a plugin to communicate to a different Data Source can be created. An Application Server that responds to HTTP requests is one of the system components. Since the Application Server is replicable, it adapts the application to the data center response needs. A separate server manages the data about the machines' performance. The data is saved on a database since the Data Source is not prepared to receive hundreds of concurrent requests. With this server, the Application Server can retrieve the data from the database, making the application replication easier. Therefore, the application can be used by any data center to exhibit information about their data center to the general public.

To exhibit the information from the data center, two scenarios are developed. Besides these two scenarios, the system allows the creation of a plugin to expose other scenes on the application. By following a set of instructions, it can create a simple 3D scenario. From this scenario, any 3D scene desired can be developed. The framework chosen to develop these scenarios is Three.js. Nevertheless, despite Three.js being a friendly framework, creating a 3D world may require a considerable amount of work. Since the application allows several

scenes, this can reach more people since it can exhibit multiple scenarios. This plugin makes the application flexible on the front-end.

To validate the application usage, the application is tested with data from a real data center. The application exposes the information successfully. Finally, the Application Server is tested to analyze its limits. A system with a single Application Server can handle a significant workload, as the performance tests show.

7.2 PROSPECT FOR FUTURE WORK

One point that could increase the diversity of the users is to create another plugin that enables the administrator to insert arbitrary information on the database. Besides, multiple scenarios could be created for this application to perform formal experiments to test the best ways to exhibit data to people.

BIBLIOGRAPHY

R. Brath, M. Peters, and R. Senior. Visualization for Communication: The Importance of Aesthetic Sizzle. In *Ninth International Conference on Information Visualisation (IV'05)*, pages 724–729, London, England, 2005. IEEE. ISBN 978-0-7695-2397-2. doi: 10.1109/IV.2005.145. URL http://ieeexplore.ieee.org/document/1509153/.

Stuart Card, Jock Mackinlay, and Ben Shneiderman. *Readings in Information Visualization: Using Vision To Think*. Academic Press, 1999. ISBN 978-1-55860-533-6.

Chun-houh Chen, Wolfgang Karl Härdle, and Antony Unwin. *Handbook of Data Visualization*. Springer Science & Business Media, 2008. ISBN 978-3-540-33036-3. doi: 10.1007/978-3-540-33037-0.

David McCandless. The beauty of data visualization - David McCandless. Available at: https://www.youtube.com/watch?v=5Zg-C8AAIGg&t=5s. Accessed: 2021-12-19.

P.W. Fach and T. Strothotte. Cognitive maps: A basis for designing user manuals for direct manipulation interfaces. In M.J. Tauber, D.E. Mahling, and E. Arefi, editors, *Cognitive aspects of visual languages and visual interfaces*, page 89–117. Elsevier, New York, 1994.

FCT | FCCN. RNCA. What is the National Network for Advanced Computing. Available at: https://www.fccn.pt/en/noticias/conheca-rnca/. Accessed: 2021-12-19.

Stephen Few. Tapping the Power of Visual Perception. *Intelligent Enterprise*, 2004.

Hwaiyu Geng. *Data Center Handbook*. John Wiley & Sons, 2014. ISBN 9781118436639. doi: 10.1002/9781118937563.

Google Data Arts Team. dat.GUI. Available at: https://github.com/dataarts/dat.gui. Accessed: 2021-12-19.

Grafana Labs. Grafana: The open observability platform. Available at: https://grafana.com/. Accessed: 2021-12-19.

Information is Beautiful. Peak Break-Up Times On Facebook. Available at: ttps://informationisbeautiful.net/2010/peak-break-up-times-on-facebook/. Accessed: 2021-12-24.

Inspari. Leveraging comparison in a stacked bar chart. Available at: https://inspari.dk/blog/da/blog/2013/12/17/leveraging-comparison-in-a-stacked-bar-chart. Accessed: 2021-12-24.

Pourang Irani and Colin Ware. The Effect of a Perceptual Syntax on the Learnability of Novel Concepts. In *Proceedings of the Eighth International Conference on Information Visualisation*, pages 308 – 314, 2004. ISBN 0-7695-2177-0. doi: 10.1109/IV.2004.1320162.

Mark Otto Jacob Thornton. Bootstrap. Available at: https://getbootstrap.com/. Accessed: 2021-12-19.

M.B. McGrath and J.R. Brown. Visual Learning for Science and Engineering. *IEEE Computer Graphics and Applications*, 25(5):56–63, 2005. ISSN 0272-1716. doi: 10.1109/MCG.2005.117. URL http://ieeexplore.ieee.org/document/1510540/.

R. Mili and R. Steiner. Software engineering—introduction. In *Revised Lectures on Software Visualization, Int'l Seminar*, pages 129–137,. 2002.

NetApp. What is High Performance Computing | NetApp. Available at: https://www.netapp.com/data-storage/high-performance-computing/what-is-hpc/. Accessed: 2021-12-24.

nuPSYS. nuVIZ 3d Visualization Data Center Infrastructure. Available at: https://www.nupsys.com/nuviz.html. Accessed: 2021-12-19.

Ken Perlin. An image synthesizer. *SIGGRAPH '85 Conference Proceedings. July, 1985. vol. 19 ; no. 3: pp. 287-296 : ill. includes bibliography*, 19, 1985. doi: 10.1145/325334.325247.

Sunbird Inc. DCIM - Data Center Infrastructure Management Software System, Cable Management, Infrastructure Design & Optimization Companies - Sunbird DCIM. Available at: https://www.sunbirddcim.com/. Accessed: 2021-12-19.

Alfredo Teyseyre and Marcelo Campo. An overview of 3d software visualization. *IEEE transactions on visualization and computer graphics*, 15(1):87–105, 2009. ISSN 1077-2626. doi: 10.1109/TVCG.2008.86. URL http://ieeexplore.ieee.org/document/4564449/.

Shaun Thomas. *PostgreSQL 12 High Availability Cookbook*. Packt Publishing Ltd, Birmingham, third edition edition, 2020. ISBN 978-1-83898-485-4.

Three.js. Three.js – JavaScript 3D Library. Available at: https://threejs.org/. Accessed: 2021-12-19.

Edward R. Tufte. *The Visual Display of Quantitative Information*. Graphics Press, 2001. ISBN 978-1-930824-13-3.

Colin Ware. *Information Visualization: Perception for Design: Second Edition*, pages 1–537. Elsevier, 2004.

Claus O. Wilke. *Fundamentals of Data Visualization*. O'Reilly Media, Inc, 2019. URL https://clauswilke.com/dataviz/aesthetic-mapping.html.

A

SUPPORT WORK

B

DETAILS OF RESULTS

C

LISTINGS

D

TOOLING

To anyone using FLASK should consider having a look at Flask web site and follow a tutorial in The-Flask-Mega-Tutorial

To Learn how to make projects on Three.js and follow examples Three.js.org is a good place to start