João Alexandre Santos Cruz

Analytics - Customized dashboards representing a descriptive analysis and diagnosis of business and operational information

Internship Report in the context of the Master in Informatics Engineering, specialization in Software Engineering, advised by Célio Gomes de Abreu and presented to the Department of Informatics Engineering of the Faculty of Sciences and Technology of the University of Coimbra.

July of 2023

This page is intentionally left blank.

1 2 9 0

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE Đ
COIMBRA

DEPARTMENT OF INFORMATICS ENGINEERING

João Alexandre Santos Cruz

Analytics - Customized dashboards representing a descriptive analysis and diagnosis of business and operational information

Internship Report in the context of the Master in Informatics Engineering, specialization in Software Engineering, advised by Célio Gomes de Abreu and presented to the Department of Informatics Engineering of the Faculty of Sciences and Technology of the University of Coimbra.

July 2023

This page is intentionally left blank.

Acknowledgements

I would like to express my sincere gratitude to Filipa Ferreira and Célio Abreu for their invaluable help, advice, support, and encouragement throughout the duration of my internship. Their guidance was instrumental in shaping my understanding and enhancing my skills. I would also like to extend my appreciation to Professor Filipe Araújo for his assistance during our meetings and his contributions to the report.

I am immensely grateful to the entire AlticeLabs ABC team for providing me with the necessary feedback and tools that significantly contributed to the improvement of my project.

Furthermore, I want to acknowledge my parents and brother for their unwavering support, allowing me to reach where I am today. I am truly thankful to my girlfriend for her invaluable assistance and for standing by me despite all the challenges. Lastly, I would like to express my heartfelt appreciation to all my friends who have accompanied me on this journey over the past five years.

This page is intentionally left blank.

Abstract

In the realm of unified communications, a substantial amount of data is generated, and service operators seek to monitor their communication networks for various purposes. The challenge lies in presenting this data in an easily comprehensible manner that facilitates actionable insights. AlticeLabs addresses this challenge through their Unified Communication as a Service, called Advanced Business Communications, by providing users with a static dashboard. However, this dashboard fails to cater to the diverse needs of different teams. To address this issue, AlticeLabs aims to develop a dynamic dashboard creation tool that allows users to easily customize widgets according to their specific requirements, using an intuitive and user-friendly interface. This internship involved researching existing dashboard creation tools and ABC competition to verify best features to add to the application, exploring best practices for dashboard design in order to give the users an easy to use and easy to design dashboard creation tool that won't allow to create wrong or misleading dashboards, identifying functional and non-functional requirements through the use of personas, and designing the architecture, followed by the development of a full-stack application and testing it as well as conducting usability testing. The expected outcome of this project is to empower users with a flexible and accessible tool to monitor their communication networks effectively and easily and make informed decisions.

Keywords

Dynamic Dashboard, Analytics, Unified Communications, Widgets, Usability, User Experience

This page is intentionally left blank.

Resumo

No domínio das comunicações unificadas, é gerada uma quantidade substancial de dados e os operadores de serviços procuram monitorizar as suas redes de comunicações para vários fins. O desafio reside na apresentação destes dados de uma forma facilmente compreensível, que facilite a obtenção de conhecimentos accionáveis. A AlticeLabs aborda este desafio através da sua Comunicação Unificada como Serviço, denominada Comunicações Empresariais Avançadas, fornecendo aos utilizadores um painel de controlo estático. No entanto, este painel não consegue satisfazer as diversas necessidades das diferentes equipas. Para resolver este problema, a AlticeLabs pretende desenvolver uma ferramenta de criação de dashboards dinâmicos que permita aos utilizadores personalizar facilmente as métricas de acordo com os seus requisitos específicos, utilizando uma interface intuitiva e de fácil utilização. Este estágio envolveu a pesquisa de ferramentas de criação de dashboards existentes e a competição ABC para verificar as melhores características a adicionar à aplicação, explorando as melhores práticas para o design de dashboards de forma a dar aos utilizadores uma ferramenta de criação de dashboards fácil de usar e fácil de desenhar que não permita criar dashboards errados ou enganadores, identificando requisitos funcionais e não funcionais através da utilização de personnas, e desenhando a arquitetura, seguida do desenvolvimento de uma aplicação full-stack e testando-a, bem como realizando testes de usabilidade. O resultado esperado deste projeto é dotar os utilizadores de uma ferramenta flexível e acessível para monitorizar as suas redes de comunicação de forma eficaz e fácil e tomar decisões informadas.

Palavras-Chave

Dashboards Dinâmicos, Análise, Comunicações Unificadas, Widgets, Usabilidade, Experiencia de Utilizador

This page is intentionally left blank.

Contents

This page is intentionally left blank.

Acronyms

ABC Advanced Business Communications.

AI Artificial Intelligence.

BI Business Intelligence.

DDI Direct Dial-In.

ECS Enterprise Collaboration Solution.

EIS Executive Information Systems.

FUD Fear, Uncertainty, Doubt.

ISO International Organization for Standardization.

IVR Interactive Voice Response.

JWT JSON Web Tokens.

MCV Multiple Coordinated View.

MoSCoW Must have, Should have, Could have and Won't have.

MVP Minimum Viable Product.

SaaS Software as a service.

SDR Service Detail Record.

SUS System Usability Scale.

SVN Subversion.

UC Unified Communications.

UCaaS Unified Communications as a Service.

UI User Interface.

UX User Experience.

This page is intentionally left blank.

List of Figures

This page is intentionally left blank.

List of Tables

Chapter 1

Introduction

This report is part of a curricular internship in Software Engineering specialization of the Master's Degree in Informatics Engineering, from the Faculty of Sciences and Technology, at the University of Coimbra, under the orientation of Célio Abreu and Filipa Ferreira of AlticeLabs and Professor Doutor Filipe Araújo, in the curricular year 2022/2023. A grant was provided by Inova-Ria program called Genius, to finance the internship and cover the expenses of trips to AlticeLabs.

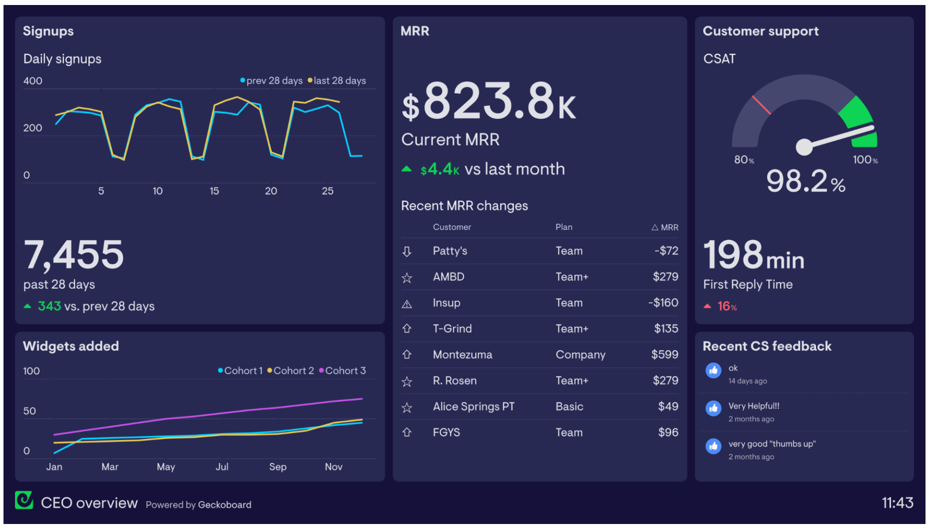
1.1 Context

To thrive in the ever-changing business landscape, enterprises must embrace adaptation and evolution. The need for tools that support collaborative work increased with the COVID-19 pandemic [59]. Enterprise Collaboration Solution (ECS) are seen as a critical enabler of the digital workspace [59], as they have been linked to increased employee productivity [23], improved collaboration, lower costs, and greater flexibility. ECS can also promote better engagement, as employees can participate from multiple platforms. Unified Communications (UC) is the foundation of ECS. According to Picot et al. [5], UC refers to the integration and management of communication methods such as messaging, voice, audio, web and video conferencing, and non-real-time communication services, in order to enhance the user experience [23]. UC platforms offer a consolidated solution that combines the advantages of Interactive Voice Response (IVR), call routing and multiple more additional features. These platforms provide unified messaging, enabling users to access various communication channels from a single interface. They also offer presence and availability features to facilitate seamless collaboration.

AlticeLabs' Advanced Business Communications (ABC) offers telecom operators, like MEO, an analytics feature for effective management of their hosted Communications Platform. Operators can monitor waiting queues, analyze user behavior, and gain insights into service usage patterns. This empowers operators to optimize performance, enhance customer satisfaction, and make data-driven decisions.

1.2 Motivation

Dashboards are an efficient way to track and consolidate data from multiple sources, providing a visual representation of the data through widgets. A widget is composed of a chart, filters and more features that allow the user to have a better experience visualizing his/her data. Dashboards offer a comprehensive overview promoting transparency and providing a single source of truth. Real-time monitoring reduces the time spent on analysis and eliminates long lines of communication that can hinder business performance. Figure 1.1 is an example of a dashboard with five widgets.



Recent MRR changes

	Customer	Plan	△ MRR
⇩	Patty's	Team	-$72
☆	AMBD	Team+	$279
⚠	Insup	Team	-$160
⇧	T-Grind	Team+	$135
⇧	Montezuma	Company	$599
☆	R. Rosen	Team+	$279
☆	Alice Springs PT	Basic	$49
⇧	FGYS	Team	$96

Recent CS feedback

ok
14 days ago

Very Helpful!!
2 months ago

very good "thumbs up"
2 months ago

Figure 1.1: Example of dashboard, taken from Geckodasboard

Currently, the ABC platform offers a static dashboard where all the widgets are created and mantained by the ABC team, meaning that operators cannot modify them or tailor them to their specific needs. By adopting dynamic dashboards that can be customized for different scenarios and situations, the ABC will give operators the flexibility to create different dashboards for each team and personnel, that have different requirements and needs. This customization capability allows operators to address individual concerns and preferences, offering a more personalized and effective solution.

1.3 Objectives

This internship aims to design, develop and implement a dynamic dashboard tool that enhances the functionality and user experience of the ABC platform and that is easy to use. The tool will empower users to create custom dashboards and enable them to monitor and analyze different aspects of their data in a more efficient and intuitive way.

The team conducted a thorough evaluation of different options to achieve this goal. After considering various alternatives, such as using a Software as a service (SaaS) Business Intelligence (BI) tool with a REST API integration, it was decided to adopt a full stack development approach. This approach will involve creating both the front-end application and the REST API from scratch. The rationale behind this decision is that this approach will provide more flexibility and control over the final product, and will also be more cost-effective, as the costs associated with using a SaaS BI tool were deemed too high for the number of users required.

The goal of this internship is to deliver a Minimum Viable Product (MVP) that features the core functionalities of a dashboard creation platform focused on usability in order to meet the needs of the users and improve the ABC platform's functionality and experience. These functionalities include the ability to create custom dashboards, add and customize widgets and filter data. This MVP will be used as the foundation for future development and will be used as a reference for the final product that will have all the functionalities of the MVP as well as the rest of the not implemented requirements specified.

1.4 Results

At the end of the internship these were the results obtained:

- Research on the state of the art of dashboards and analysis of ABC competitors. A comprehensive review of existing dashboard solutions and an examination of competitors' offerings were conducted;
- A detailed requirements list was created, including personas, use cases, and non-functional requirements;
- Architechtural drivers that allow for future development;
- All must-have and should-have functional requirements were implemented, documented, reviewed and tested;
- A usability questionnaire was designed and administered to measure the usability of the application;

- The developed application was successfully deployed, making it accessible for testing and evaluation.

The internship culminated in the development of a MVP that successfully meets the specified requirements and adheres to the defined constraints. This includes the implementation of an API to facilitate communication with the frontend application, which was also developed as part of the internship.

The MVP not only encompasses all the required functionalities but also incorporates additional features that were categorized as should-haves.

While the developed application meets the immediate needs and objectives of the project, it is acknowledged that more extensive testing could have been conducted to ensure its robustness and reliability. Furthermore, there is room for future work and improvements on the application. Areas such as performance optimization, user interface enhancements and additional features can be explored and developed in subsequent iterations. The internship has laid the foundation for future development, and the identified areas for improvement provide a clear roadmap for further enhancing the application's functionality and user experience.

1.5 Document Structure

The structure of this report is divided into the following chapters:

- Chapter 2 focuses on the work plan for the project. It provides charts and descriptions of the schedule for the first semester and expectations for the second semester. This chapter also conducts a risk analysis to document potential obstacles to the project;
- Chapter 3 is dedicated to the state of the art and background. It begins with a brief clarification of the concepts relevant to the project. After conducting an extensive analysis of UCaaS platforms and dashboards, the selected technologies are presented with an explanation of why they were chosen;
- The introduction of architectural drivers is done in Chapter 4. It presents and explains functional and non-functional requirements, use cases and C4 models;
- Chapter 5 of the report provides a detailed explanation of the implementation process and the methodology followed throughout the project. This chapter not only presents the methodology used but also describes all the endpoints that were developed for the API, as well as the components and pages that were developed for the frontend of the application. It serves as a

comprehensive guide to understanding the technical aspects and the functionality of the implemented system;

- Chapter 6 of the report presents an overview of the tests conducted on both the API and the frontend of the application. This chapter provides detailed information about the test scenarios, test cases, and test results for both the API endpoints and the frontend components. Chapter 6 also presents the results of the usability questionnaire that was conducted. The chapter includes an analysis of the questionnaire responses and provides insights into the usability of the application based on the participants' feedback;

- Chapter 7 of the report serves as the conclusion, providing a comprehensive summary of the internship experience, including an analysis of what went well and what did not meet expectations. This chapter offers an opportunity for reflection and a discussion of the lessons learned during the internship. This chapter also explores the future work and potential improvements that could be made to the project.

This page is intentionally left blank.

Chapter 2

Work Plan

In this section, the proposed work plan for both semesters is presented and explained.

2.1 First Semester

This chapter presents and discusses the expected planning and the actual plan for the first semester.

Planned Schedule

The planning for the first semester was done by the supervisors at AlticeLabs. The first semester was a part-time with 16 hours scheduled per week. The original plan included researching the state of the art, followed by the requirements and system architecture, and finally writing the mid-term report. In the figure 2.1 the gantt chart for the planned work plan is showed .

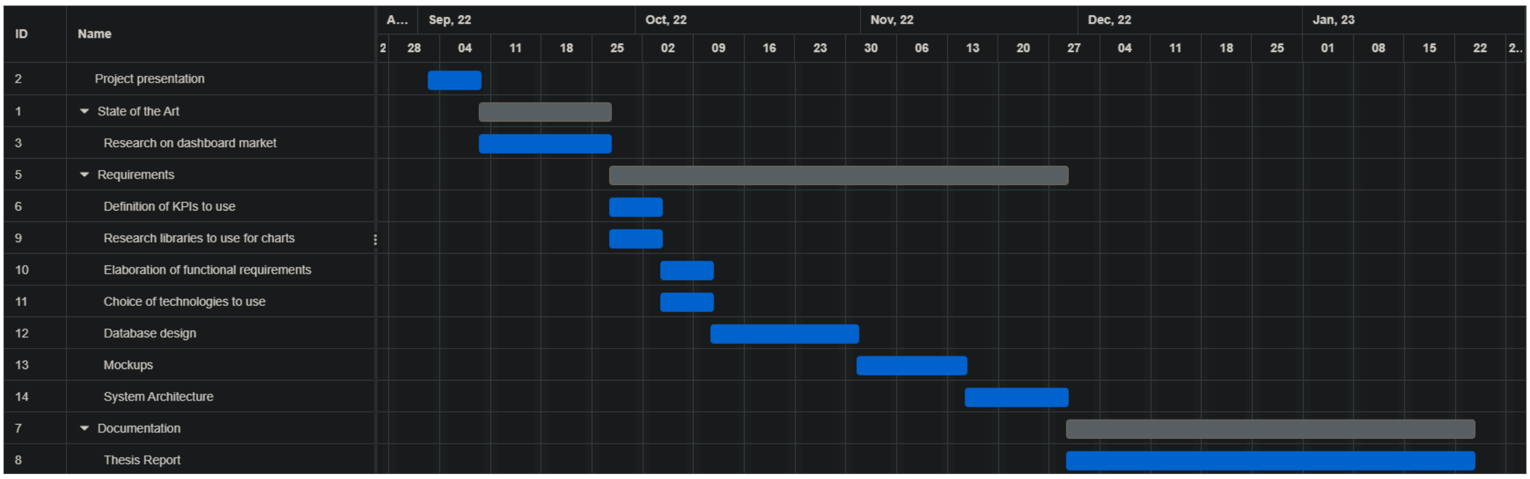


Figure 2.1: Gantt chart with the planned schedule for the first semester

Actual Schedule

The actual schedule underwent significant changes compared to the initially expected timeline. In the first semester, regular feedback received during team meetings played a vital role in shaping the project's direction. These meetings provided valuable opportunities for discussions, particularly regarding the integration of a BI SaaS tool with a backend application. As depicted in Figure 2.2, additional time was dedicated to research, resulting in delays for other planned activities. To mitigate potential risks, the database design phase was also postponed.

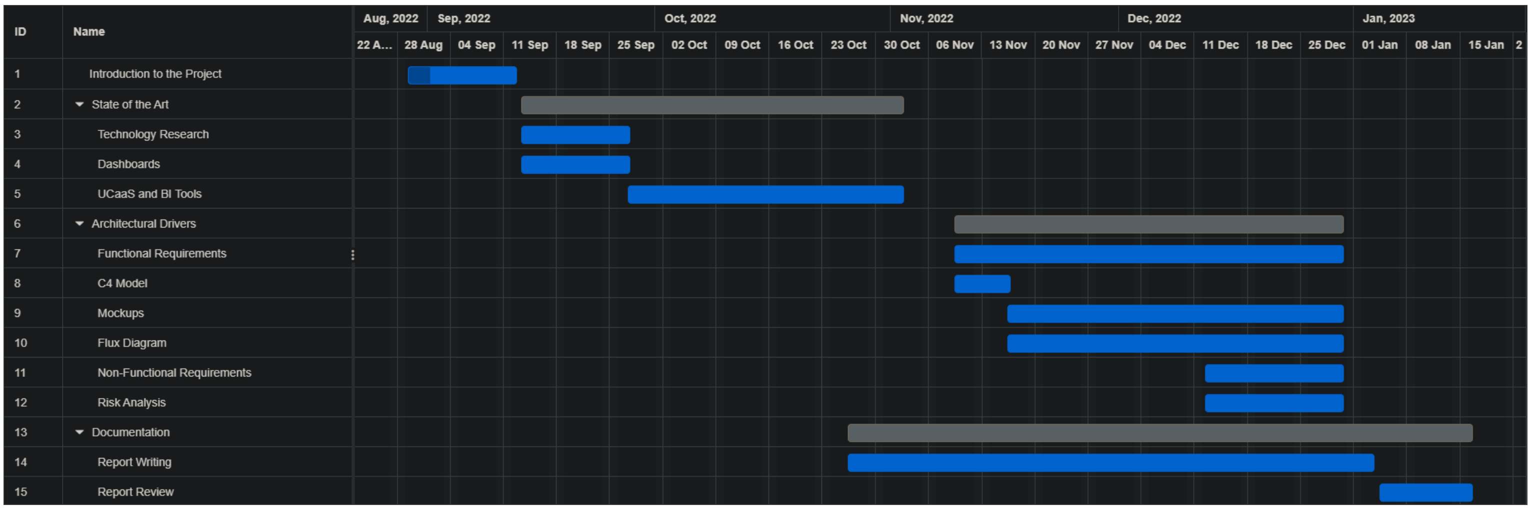


Figure 2.2: Gantt chart with the actual schedule for the first semester

2.2 Second Semester

This chapter presents an overview of the planned timeline the projected schedule for the second semester.

Planned Schedule

Following the mid-term delivery, the planning for the second semester was undertaken by the supervisors at AlticeLabs, as depicted in Figure 2.3. The proposed approach was to follow a waterfall methodology, encompassing the completion of requirements, development of APIs, and subsequent frontend development. The indicated development time includes not only the coding phase but also documentation and review processes. A testing phase, along with an accompanying bug fixing phase, was also incorporated into the planned timeline.

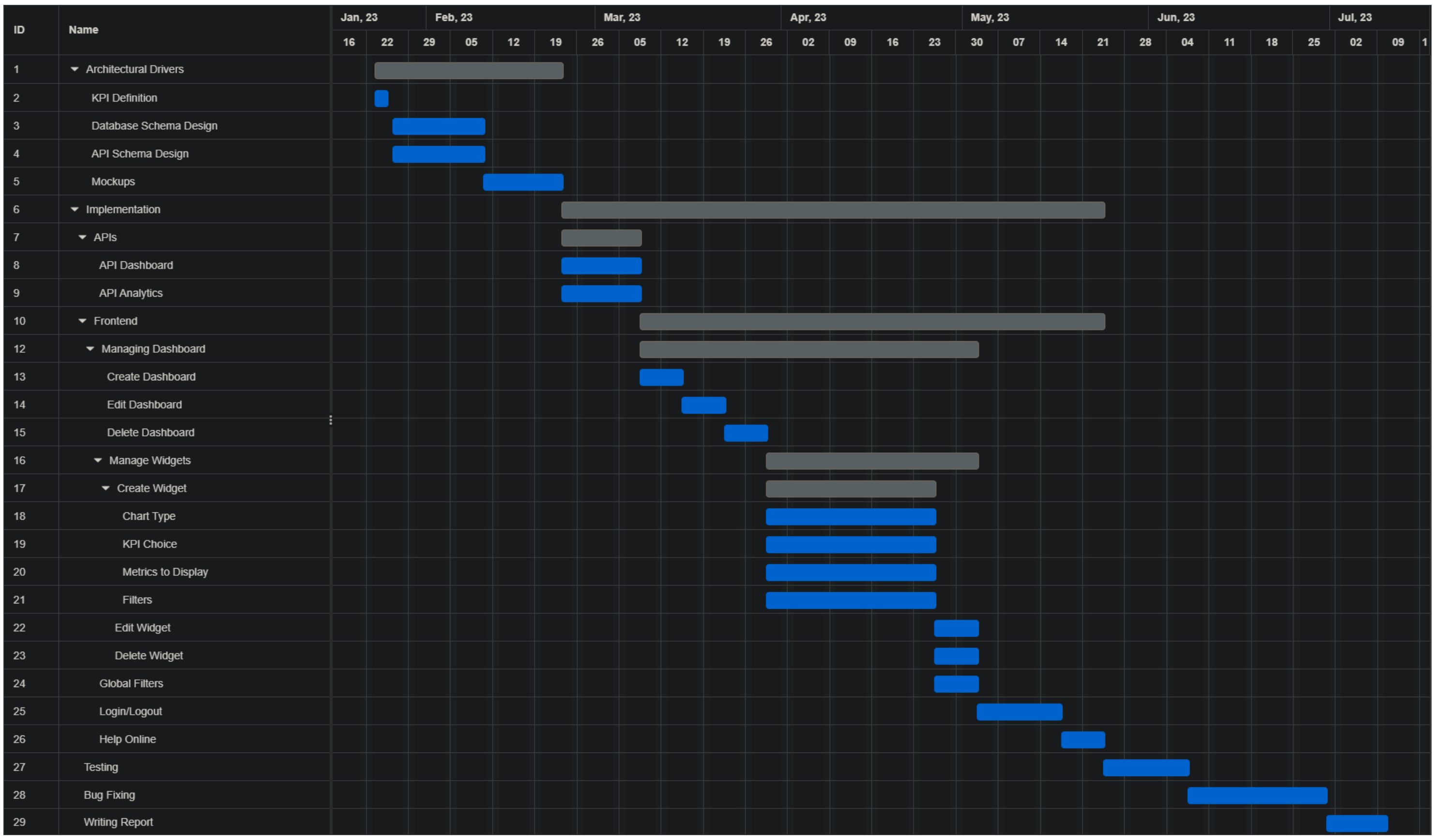


Figure 2.3: Gantt chart with the planned schedule for the second semester

Actual Schedule

The second semester witnessed even more changes to the schedule, as showed in figure 2.4. A shift in methodology occurred, transitioning from the waterfall approach to the iterative and incremental methodology. Moreover, more features were developed than initially anticipated. Consequently, the database design, API schema, and mockups were not fully defined at the project's inception, and these elements underwent modifications during each increment.

The increased scope of development and the introduction of new features necessitated additional development time. Furthermore, several challenges arose, which will be discussed in chapter 7. Testing and bug fixing commenced not only at the end of each increment instead of only at the end.

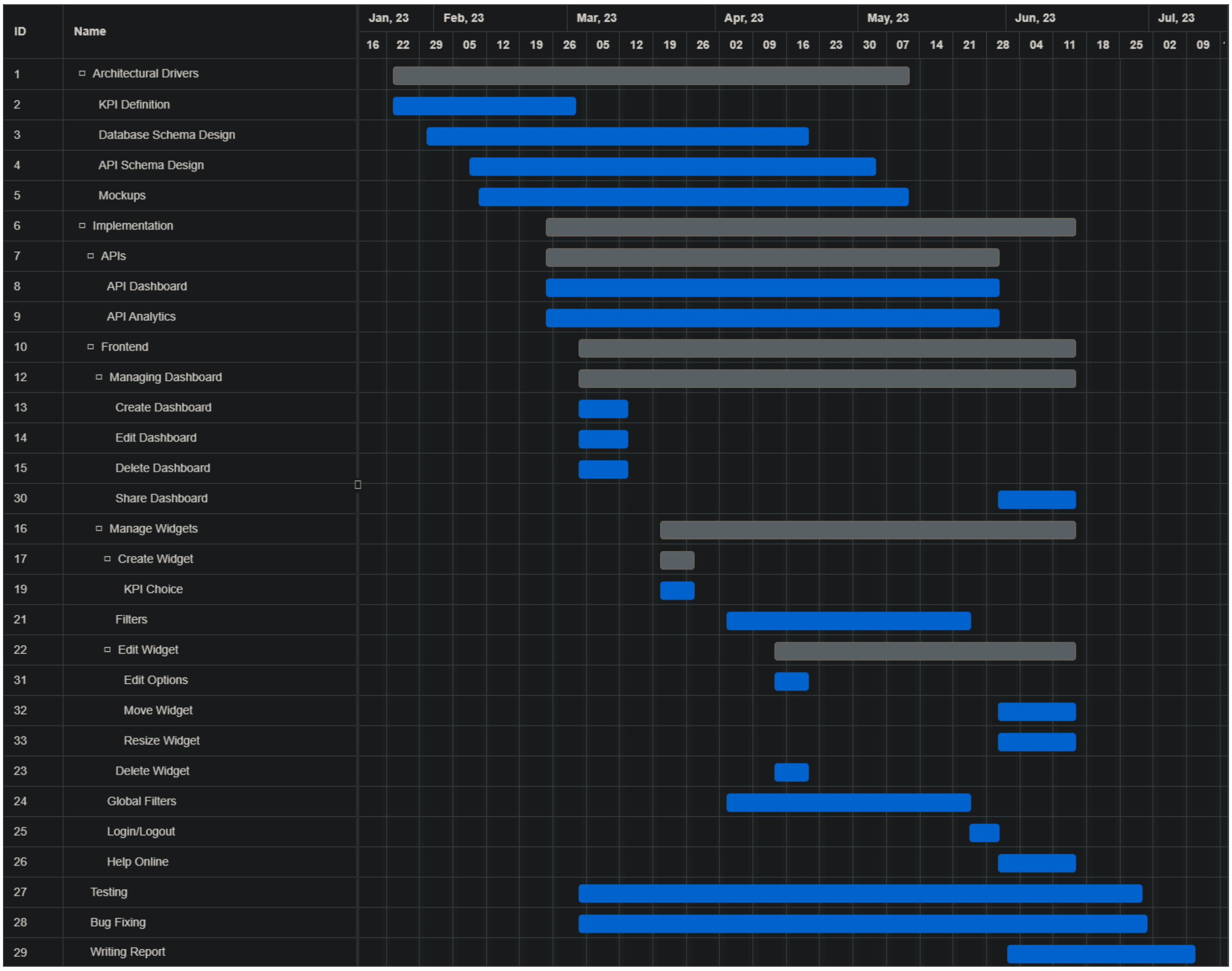


Figure 2.4: Gantt chart with the actual schedule for the second semester

2.3 Risk Assessment

Risk analysis is an important part of any project, as it helps to identify and evaluate potential risks that may impact the project's success. By understanding the potential risks it's possible to take proactive measures to mitigate or eliminate them, increasing the likelihood of project success. In this section, the risks identified in this project are going to be defined.

2.3.1 Threshold of Success

The threshold of success defines what's considered as an accomplished project, if any goal of this threshold is not accomplished then the project should be considered as a failure. These goals were defined as a team.

- The *Must Have* functional requirements specified should be developed, reviewed, documented, tested and be ready to deploy for production;
- The developed system must comply with the non-functional requirements specified;
- The developed system must comply with the restrictions.

To evaluate the success threshold and determine the achievement of objectives, clear objectives were set. The requirements were divided into tasks, and upon completing each task, a team member reviewed it. The team agreed that the documentation should include details about each endpoint as well as each frontend component.

Regarding the tests, it was determined that API testing should adhere to the non-functional requirement, as specified. On the other hand, frontend testing would solely involve manual testing, with a focus on the integration with the API. For further information regarding the tests, please refer to Chapter 6.

2.3.2 Risks

The risks identified where:

1. As the platform cannot be completely defined, the database will undergo changes during development. Since the database is being developed in collaboration with another team at AlticeLabs, any necessary changes will delay the development process;
2. The proposed mockups may be impossible to replicate, requiring changes to the front-end design that will need to be reevaluated by the team, causing delays
3. Since I don't have experience on front-end development, it is possible that front-end development may take longer than expected
4. Since I don't have any experience using automated testing tools, testing may take longer than expected

To determine the level of risk for each identified risk, the 5x5 risk matrix was used. The 5x5 risk matrix is a tool that helps to evaluate and prioritize risks based on their likelihood and impact. It is easy to use and understand, and can be easily incorporated into the project management process. This tool allows to focus on the most significant risks and allocate resources accordingly, the matrix is display in 2.5.

Table 2.1 presents the mitigation plan as well as the severity, likelihood and risk level of the risks mentioned above.

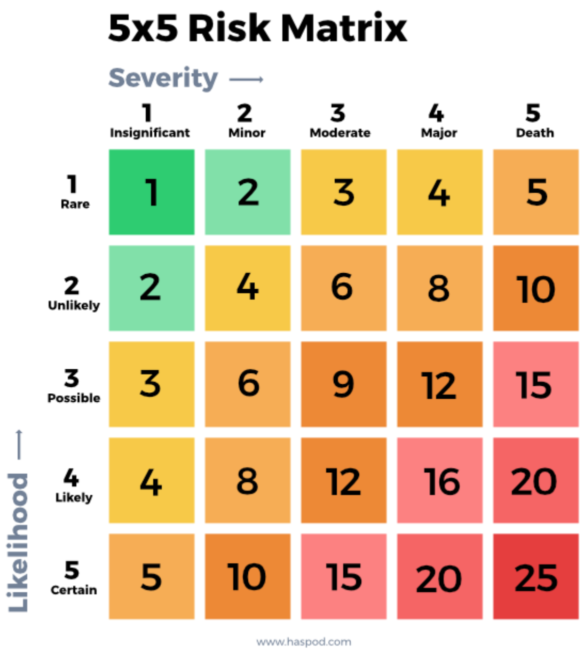


Figure 2.5: 5x5 Risk Matrix

Risk	Mitigation Plan	Severity	Likelihood	Risk Level
1	To reduce the number of times changes are required, the database design should be delayed until most of the details are documented	4	5	20
2	To ensure the feasibility of the mockups, observe how other similar tools are designed	2	3	6
3 and 4	Use the beginning of the semester to watch tutorials and the trips to AlticeLabs to learn from experienced personnel	3	5	15

Table 2.1: Comparison of the different dashboard types

Regarding the first risk, it was not possible to completely mitigate it. The database did undergo changes with each increment, resulting in delays during each iteration.

Risk two was successfully mitigated. The implemented screens faithfully followed the mockups, indicating that the mitigation plan of using similar tools for inspiration was effective.

Risk three can be considered mitigated. Although the initial work did not progress at the expected pace, the subsequent tasks compensated for the initial delay. In fact, additional tasks were even accomplished that were not initially planned.

Risk four was successfully mitigated by conducting automated testing on the API as expected. However, it should be noted that the risk also encompassed the need for automated tests on the frontend, which was not carried out in compliance with AlticeLabs QA guidelines, more details are showed on chapter 6.

This page is intentionally left blank.

Chapter 3

Background and State of the Art

Dashboards have become an integral part of modern business operations, providing a real-time view of key performance indicators and enabling organizations to quickly make data-driven decisions [6]. As technology has advanced, so too have dashboards, becoming increasingly sophisticated and user-friendly. This chapter delves deeper into the state of the art and background of dashboard systems, exploring the current trends, innovations, and challenges in the field. By gaining a comprehensive understanding of these aspects, it's possible to design and develop a dynamic dashboard tool that effectively meets the needs of the ABC platform and allows its users to create good and effective dashboards.

3.1 Dashboard

Dashboards have been utilized by businesses to analyze and interpret data since the 1970s. With the growth of big data, dashboards have evolved to become more user-friendly [26]. The development of dashboards can be traced back to Executive Information Systems (EIS), which were initially designed for executive offices and showed only a limited number of crucial financial metrics through a straightforward interface for easy understanding. In the 1990s, Robert Kaplan and David Norton [47] introduced the Balanced Scoreboard, which marked a change in approach by utilizing KPIs for the first time. The Enron scandal in 2001 [40] was a turning point for dashboards, as corporations needed to demonstrate to their shareholders their ability to closely monitor performance.

According to Few [26], dashboards are predominantly visual information displays that people use to quickly monitor current conditions that require a timely response to fulfill a specific role, later Few also added that dashboard should be "consolidated and arranged on a single screen so the information can be monitored at a glance" [27]. Wexler et al. [73] offers a broader definition: "a visual display of data used to monitor conditions and/or facilitate understanding", which means that in addition to conventional dashboard displays, there is the possibility of incorporating infographic elements or narrative visualizations.

This section delves into the types of dashboards, the role of audience in dashboard design, the selection of appropriate graphs for various data, and the tools utilized by BI teams for dashboard creation.

3.1.1 Effective Dashboards

When creating a dashboard, it's important to first determine what questions the dashboard should answer. Based on this, effective visual encodings should be chosen for graphical features such as position, size, shape, and color [37]. The aim of visualization is to enhance understanding of data by leveraging the human visual system, which is excellent at detecting patterns, identifying trends, and uncovering outliers.

Although Business Intelligence vendors have advanced in the gathering, transforming, and storing of data, there have been limited advancements in the proper utilization of this information. As will be discussed later, an effective dashboard results from effective communication [26] rather than just good data. As R. Barth and M. Peters [10] point out a dashboard without good visual information is of no use:

> Dashboards and visualization are cognitive tools that improve your "span of control" over a lot of business data. These tools help people visually identify trends, patterns and anomalies, reason about what they see and help guide them toward effective decisions. As such, these tools need to leverage people's visual capabilities. With the prevalence of scorecards, dashboards and other visualization tools now widely available for business users to review their data, the issue of visual information design is more important than ever.

3.1.2 Dynamic vs Static

Bach et al. [9] classified dashboards into six genres, with two of them being the most common: static and dynamic (or analytic). Static dashboards are uniform and unchanging for all viewers, with data being displayed consistently. These dashboards generally offer limited interaction options for the user and the data updates automatically. On the other hand, dynamic dashboards can be tailored to individual user preferences, allowing them to choose the KPIs to view and how they want to view them, and providing features such as drill-downs [1] and interactive filters. While static dashboards are still commonly used, there is a growing

[1]Technique or functionality that allows users to explore detailed or granular information by progressively navigating from a high-level summary view to a more specific or detailed level of data.

trend towards dynamic dashboards as the importance of User Experience (UX) increases in clients' search for tools.

Dynamic Dashboards

Dynamic Dashboards have evolved from Multiple Coordinated View (MCV). J. C. Roberts defined MCV [54] as "computer systems that simultaneously display and manipulate multiple perspectives of the same data, information, or objects." MCV offers a more comprehensive understanding of the data by allowing users to view and analyze it from different angles using advanced visualization and interaction techniques. However, MCV requires all views to be linked and does not support real-time data or customization. Dynamic Dashboards merge the strengths of MCV with the ability to handle real-time data and offer customization options.

J.C. Roberts also introduced the concept of linked views in his work [54], which connects multiple views in a way that any changes to one view are reflected in the others. This concept is commonly referred to as interactive filtering in dynamic dashboards. M. Frose and M. Tory [29] further divided this technique into two parts: brushing and linking. Brushing refers to the ability to select data directly from a visual representation, and the linked charts, these are charts with common data, are then automatically highlighted based on the selected data. This allows the user to manipulate the data directly and visualize the information of interest. Another common feature found on dynamic dashboards is the ability to drill-down allowing the user to gain a deeper understanding of the data and to identify patterns and relationships at different levels of detail.

Other features commonly present on dynamic dashboards have evolved from MCV. For example, dynamic queries [58, 11], which allow users to modify queries on the fly via sliders or buttons, which allows customization of colors, chart types, and other aspects of the display such as zooms and focus[49].

3.2 Roles of Dashboards

The intended use of a dashboard determines the visual design and functional requirements of the dashboard. Dashboards are no longer just decision-support tools [56]. Both Few [26] and Eckerson [24] among others divide dashboards into four roles, each of which differing from the others in terms of the amount of temporal lead and lag in decision time, the type of data presented, and the intended audience. These differences are described in the following table, table 3.1, and further explained in the following subsections.

Type of Dashboard	Level of Seniority	Time Application	Complexity of Data
Strategic	Senior Management	Long-Term	High
Tactical	Middle Management	Medium-Term	Medium
Operational	Junior Management	Daily	Simple
Social	Not Aplicable		

Table 3.1: Comparison of the different dashboard types

3.2.1 Strategy

A **strategy dashboard** monitors the long-term company strategy and displays critical success KPIs.

These dashboards are challenging to develop and provide insight into the impact of the enterprise community on the business. They are primarily utilized by senior management and often provide an overview of performance over specific periods of time, such as quarters or years. These dashboards benefit from the inclusion of contextual information and are not meant for interaction.

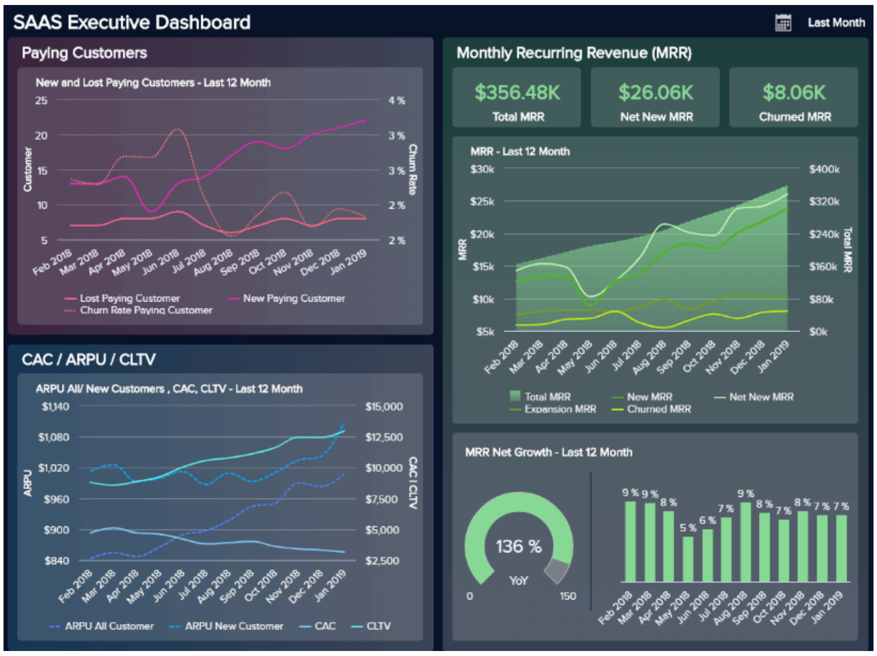


Figure 3.1: Example of a strategy dashboard from a SaaS company [16]

Figure 3.1 shows an example of a strategy dashboard for any SaaS company, with sections displaying data on customers, revenue, and costs. The left side of the dashboard shows customer gains and losses over the course of a year, enabling management to see if new strategies were successful and if defined targets were met. The right side of the dashboard displays monthly revenue, which is the most critical KPI and is given more space to make it more visible to managers.

3.2.2 Tactical

Tactical dashboards, also referred to as analytical dashboards, serve as a valuable tool for monitoring the processes that contribute to an organization's strategic initiatives and aiding in the decision-making process. They bridge the gap between operational and strategic dashboards by providing a comprehensive display of both real-time and historical data, offering detailed insights. These dashboards enhance interactivity through features like drill-downs and global filters, empowering analysts to swiftly modify all charts based on specific filters of their choice.

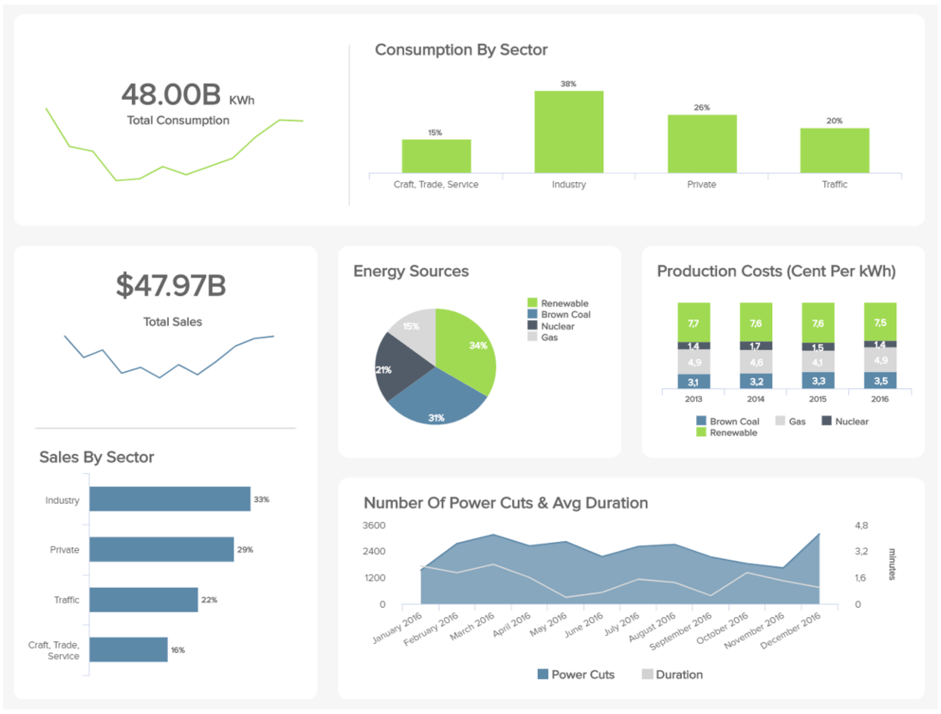


Figure 3.2: Example of a tactical dashboard from an Energy Power Plant Company [16]

Figure 3.2 shows an example of a tactical dashboard for an energy power plant company, providing management with data on total sales, consumption by sector, and production costs per source type. This allows management to see the overall performance of the company and how many clients are interested in different energy sources, helping them to decide where to invest more. The sales and consumption data by sector also enables management to know where to focus their marketing efforts and identify areas where the product may be lacking.

3.2.3 Operation

An operational dashboard is a visual display of metrics in real-time or from the recent past that can be tied to specific responsible entities and allow for immediate action in case of problems. It is designed to provide a quick overview of the day-to-day operations of a business.

Operational dashboards are typically used by team managers and analytic teams and display data that is contextualized for each department, rather than the entire enterprise, allowing them to take the appropriate approach with their team members. These dashboards often offer the ability to receive daily email reports with snapshots of the dashboard, allowing managers to compare performances and access more detailed information.

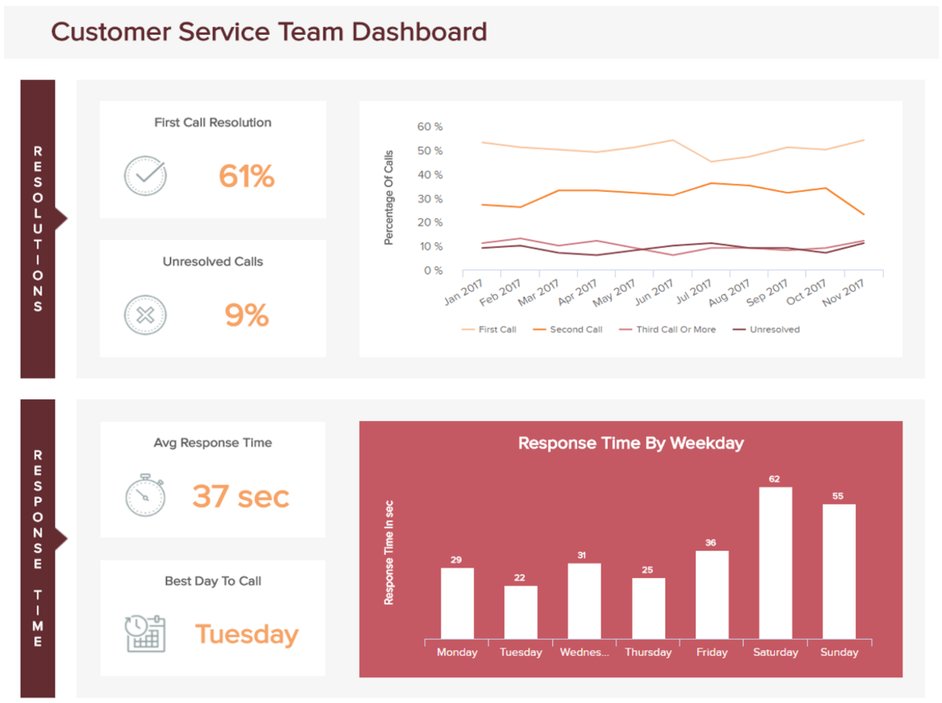


Figure 3.3: Example of an operational dashboard from a Customer Service Team [16]

Figure 3.3 shows an example of an operational dashboard for a customer service team, displaying daily and monthly performance data, including the number of calls. The dashboard is divided into two parts to allow the manager to focus on the relevant data at any given time, and the response time data by weekday allows the manager to see if more staff are needed on certain days.

3.2.4 Social

A social dashboard is a type of dashboard that is focused on informing and communicating with the reader, rather than on decision-making. This type of dashboard is particularly useful for readers who may not have the necessary context to understand the data being presented.

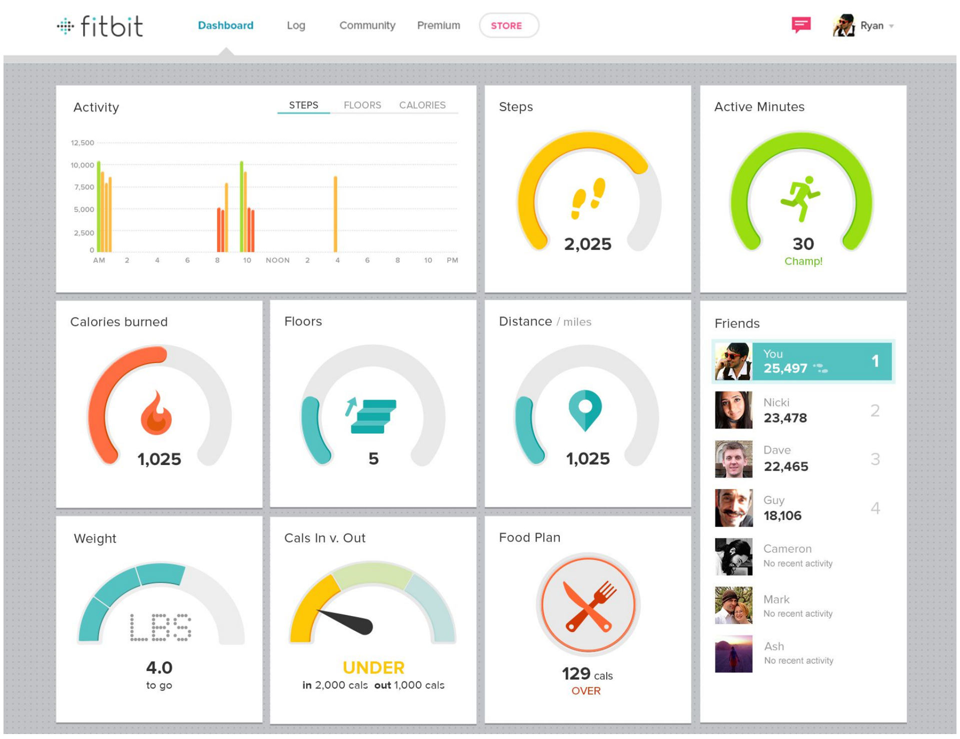


Figure 3.4: Example of a Social Workout dashboard [56]

Figure 3.4 shows an example of a social dashboard for tracking workouts, providing the user with data to help them understand the context of their workout data.

3.2.5 Audience

The intended audience of a dashboard plays a crucial role in determining its visual and functional aspects. When assessing the audience, two key parameters come into play: circulation and literacy.

Circulation

Circulation refers to the type of audience that the dashboard will be displayed to. A. Sarikaya et al. [56] identified four groups: public, social, organizational, and individual. Each group requires a different level of data context.

Public dashboards are intended for general consumption and are often used for advertising or providing information. The data presented should require no context as it needs to be general knowledge. An example of a public dashboard is one that displays crime rates by state and type. Social dashboards provide visualizations of a user's personal information, which can be shared with individuals of the user's choosing and therefore requires context from the user themselves. An example of a social dashboard is a fantasy league dashboard for a user's team, with stats for each player. Organizational dashboards are intended for use by multiple individuals with a common goal, such as a dashboard exploring the customer relationship with a business, with visualizations of states, vehicles, etc. Individual dashboards display data specific to an individual, such as a dashboard showing a user's home energy usage.

Required Visualization Literacy

The more complex a dashboard is, the more literacy it requires, as it needs to be comprehensible to the user. Low literacy dashboards are easy to see and understand, and typically use bar, line, or pie charts to display data. This makes the data easy to interpret. Medium literacy dashboards introduce more complex charts such as heatmaps, dual axes, and scatter plots, which may be more difficult to read. High literacy dashboards are intended for specialized analysts or students and may include error bars, treemaps, and other complex charts such as networks or custom charts. These dashboards require a higher level of literacy to interpret and understand the data.

3.3 Clusters of Dashboards

A. Sarikaya et al. [56] created a collection of dashboards from various sources and encoded each design decision as a character string. They used the Hamming Distance to calculate the distance between dashboards and created clusters of similar dashboards. They found 7 clusters with different goals, based on the types of decisions they supported and the visual and functional features of their design. With this they intended to provide a guideline of what features each cluster should have, dependent on audience and purpose, creating a more concise framework for dashboard creation.

The following table, 3.2, provides a summary of each cluster with a more detailed explanation provided in the following subsections.

		Purpose				Audience			Features						Semantics		
Goal	Cluster	Strategic	Tactical	Operational	Social	Audience	Literacy	Domain Expertise	Construction	Interactivity	Modify Data	Highlighting	Areas of Work	Alerts	Benchmarks	Updatability
Decision Making	Strategic Decision-Making	Y	Y	-	N	O	-	-	-	Y	N	N	Y	-	-	Y
	Operational Decision–Making	N	Y	Y	N	O	-	-	-	Y	N	N	Y	-	Y	Y
Awarness	Static Operational	N	N	Y	N	O	L	-	-	-	N	N	N	-	Y	Y
	Static Organizational	-	-	N	N	O	M	-	N	N	N	-	N	N	-	Y
Motivation and Learning	Quantified Self	N	N	Y	N	I	H	N	N	Y	N	-	Y	-	-	Y
	Communication	-	-	-	Y	P	M	N	N	-	N	-	-	N	N	Y
	Dashboards Evolved	-	-	-	-	P	H	-	-	-	-	-	-	-	-	Y

Table 3.2: Observed characteristics for each cluster of coded dashboards. **Y** indicates presence, **N** indicates the opposite, and **-** indicates that no conclusion could be reached. **O** represents organizational audiences, **I** individual, and **P** public. **L**, **M**, and **H** indicate the level of literacy, respectively low, medium, and high. Taken from [56]

By analyzing these clusters, its possible to insight into the common goals and objectives users have when creating dashboards, enabling the platform to offer the necessary tools and features to support different types of intents. This will help ensure that users can effectively and efficiently create dashboards that meet their specific needs and requirements.

3.3.1 Features and Semantics of Dashboards

Before explaining each cluster, it is necessary to provide a brief introduction to the common features and semantics found in the dashboards. This introduction will build upon the purpose and audience that were explained in the previous section.

- Construction and Composition → This functionality allows the user to select the metrics to be visualized, choose the type of charts to use, and resize and move the chart to the desired location on the dashboard;
- Areas of Work → By having multiple areas of work, users can quickly switch between dashboards to access the necessary information or context;
- Interactivity → Interactivity between charts may involve filters, slicers, global filters, and drill-downs, allowing users to focus their analysis on the relevant data;
- Annotations → Annotations allow users to highlight their opinions of a particular chart, allowing other users to examine them and facilitating collaboration;
- Alerts → In real-time data dashboards, users can set thresholds on specific charts that, when met, trigger notifications so the user can take immediate action to address the issue at hand;
- Updatability → Real-time dashboards are constantly or frequently updated with the latest data stored. This feature is not applicable to historical dashboards.

3.3.2 Dashboards for Decision-Making

These dashboards support either strategic or operational decision-making and are typically targeted at organizations. They allow for interaction to obtain and focus on relevant data for the users and often have a benchmark setting that provides visual alerts for areas of concern. These are the typical business dashboards used to understand sales in real-time (for operational purposes) or over a defined period (for strategic purposes).

3.3.3 Static dashboards for Awareness

These types of dashboards are targeted at increasing general awareness and therefore do not include interactive elements. Static operational dashboards provide real-time data from sensors or metrics and require a higher level of domain knowledge due to the additional context needed to understand the data. Static organizational dashboards are meant to be quickly consumed, such as those sent in emails.

3.3.4 Motivation and Learning

This cluster contains two types of dashboards. The first type is focused on individuals and is primarily used for tactical and operational decision-making, with interactive interfaces and alerting capabilities. These typically deal with personal matters such as finance. The second type is intended for the general public and is designed to communicate and educate the consumer.

3.4 Dashboard Patterns

B. Bach et al. extended the work of Sariakaya et al. [56], Card and Mackinlay [17], Chen [18], He et al. [33], Schulz et al. [32] and Sedig et al. [60], as all of these works focused on the intentions behind each dashboard but not on their structure and visual design. Starting with the 80 dashboards analyzed by Sariakaya et al., they added an additional 64 dashboards from various sources such as health, fitness, and IT websites and personal applications. B. Bach et al. employed a comparative method [31] to independently code the dashboards and identified 42 design patterns grouped into eight categories, divided into two groups:

- Content $\rightarrow$ focused on how the data is abstracted, meta information and visual representations used;
- Composition $\rightarrow$ page layout, information fitting, screenspace and overall structure of the dashboard.

Prior to delving into the patterns, it was necessary to establish a guideline. Drawing from previous studies [75, 26, 50, 56, 36], the following criteria were defined for the dashboards:

- Avoid overwhelming users [75].
- Steer clear of clutter [26].
- Eliminate poor visual design [50].
- Incorporate both visual and functional features [75].
- Maintain consistency with an optimal level of complexity [56].
- Enable separation of charts [36].

An analysis of these patterns can provide insight into the necessary tools and features that a platform should offer to enable users to create effective dashboards with good content and component creation capabilities.

3.4.1 Content Patterns

In the content group, there are three categories which are the main components of the dashboard. These categories determine what is visualized and how it is visualized.

Data Information

Data information identifies types of information presented:

- Individual values: represent specific data points in a data set;
- Derived values: information from datasets or KPIs;
- Filtered data: shows detailed data;
- Thresholds: judges a data point based on thresholds made by the user;
- Aggregated data: results from the junction of various data sets into a new concise data point;
- Detailed datasets: offer a more complete representation of data with a numerous of qualities and information.

Meta Information

Meta information captures additional information used to provide context and explanation to the visualization:

- Data Source: involves identifying the sources of data and understanding the methods used for its collection;
- Disclaimer: indicates what assumptions have been done to the data, serves as the context for the chart or widget;
- Description: high-level description of what the dashboard is supposed to show;
- Update Information: information of when the data was updated;
- Annotations: notes added by the dashboard author to highlight points or changes on the data.

Visual Representations

Finally, visual representations describes what solutions are used to present the information in the dashboards:

- Numbers: individual numbers placed to indicate a single key value;
- Trend-arrows: small arrows that indicate the direction of change of data values;
- Pictograms: symbols that illustrate concepts in the data, used to represent data or designate the type of the data;
- Progress bar: used to represent ranges of values, divided into thresholds;
- Signature charts: smaller visualizations with no context or description, used to give a quick understanding of smaller and precise values;
- Detailed charts: charts with enough details to read and understand precise values;
- Text Lists: lists of information by text.

3.4.2 Composition

The composition of a dashboard is every component that allows to change the display of the dashboard itself.

Layout Patterns

Layout patterns identify how the widgets are organized into the dashboard [35, 7]:

- Open Layouts: widgets laid out with no rules, user may select size and position of every widget as he pleases;
- Table Layouts: widgets aligned in columns and rows;
- Stratified Layouts: give a top-down ordering of widgets;
- Grouped Layouts: Group widgets by relation.

Screenspace

Screenspace represents the available space that can be consumed by the ammount of the data:

- Screenfit: dashboard is fully visibale on screen with all information shown at all times, doesn't provide any type interaction;
- Overflow: allows dashboard to have data off-screen that can be revealed trough scrolls, allowing a single page to be used for the whole dashboard;
- Detail-on-demand: information is represented once the user interacts on the available charts to get more details;
- Parameterized: users contol that content is showed on screen;
- Multipage: dashboard composed of various widgets distributed in various screens available trough pages, that can be accessed trough tabs.

Structure

Structure describes how the information, in case of using multiple pages, is represented among pages and influence the navigation of users:

- Hierarchical: multiple pages organized in a hierarchy that implies relationship between pages;
- Parallel: no levels and no relationship between pages;
- Semantic: relationships determined by the semantics of the information.

Interaction

Interaction approaches highlights common roles interaction plays in dashboard use through specific user interface components, defined broadly to identify general usage patterns and their implementation in dashboard designs:

- Exploration: interactions allow users to explore data relationships, includes brushing and linking and detail-on-demand interactions;
- Navigation: interactions that allow the users to navigate trough different information sources, can be trough different dashboards, scrolling, buttons or tabs that link different pages;
- Personalization: users can configure and define the dashboard;
- Filter and Focus: searching trough data visualizations.

3.5 Dashboard Tradeoffs

B. Bach highlighted the trade-offs that a user must consider when designing a new dashboard [9]. These trade-offs are depicted in figure 3.5. By analyzing these tradeoffs, it's possible to understand the various considerations that a user must make when designing a dashboard. This knowledge will allow the platform to provide all the necessary tools and features to meet the diverse needs of users.

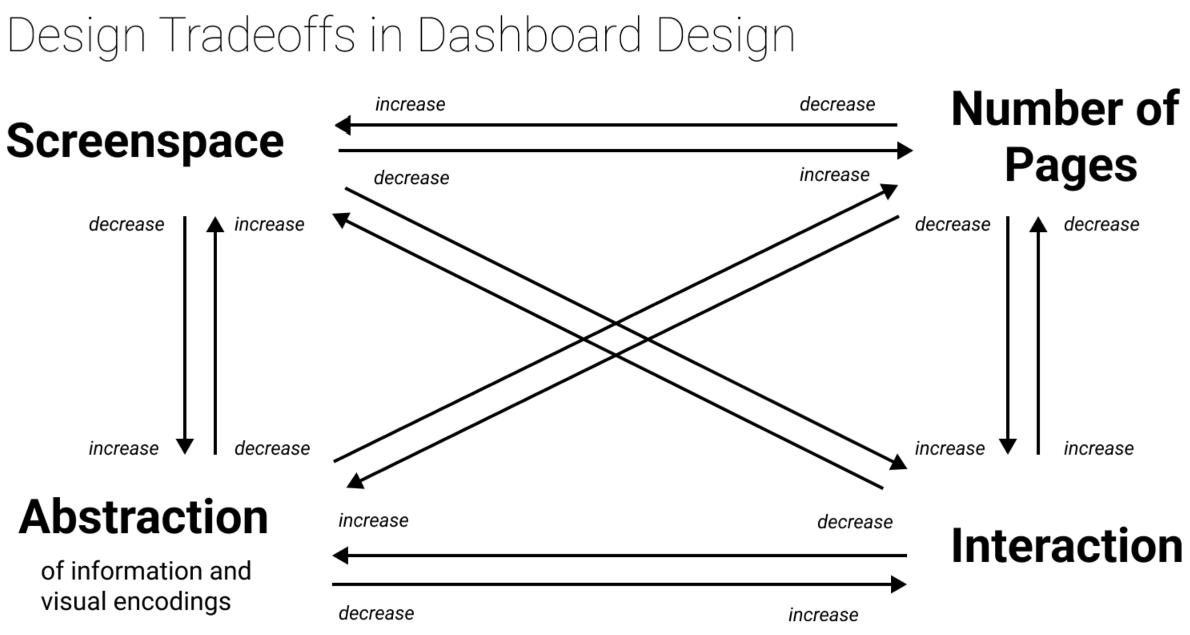


Figure 3.5: Dashboard Tradeoffs [9]

The goal is to minimize every parameter such task, however is impossible. To show all the information at a glance without any interaction, some parameters must be compromised in order for others to improve. As showed in figure 3.5 B. Bach identified four main parameters: available screen space, number of pages where the information is distributed, visual abstraction, and the level of interactivity provided to the user.

As the platform will be used on a computer screen, the amount of interactivity needs to increase to compensate, therefore the dynamic dashboard approach is necessary. The abstraction and number of pages are left to the user to decide as the platform will need to support both high and low levels of these parameters.

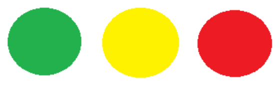
3.6 Dashboard Display

In this section, we will explore the different methods of display used in dashboards and common errors and pitfalls that occur. To effectively communicate data information to users, it's crucial to consider the display methods and be aware of the common mistakes in order to give the platform the necessary tools to minimize the chance of users creating misleading or ineffective dashboards.

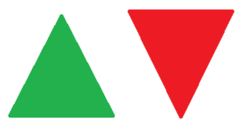
3.6.1 Icons

Icons are a powerful tool in dashboard design, as they allow for quick and easy visualization of data and information. By utilizing icons, users can efficiently and effectively communicate data insights to others. Few divides icons intro three sub-categories shown in 3.6:

(a) Example on off icons



(b) Example of alert icons



(c) Example of arrow icons

Figure 3.6: Examples of each icon sub-categories

Alerts typically use a traffic light approach, as depicted in Figure 3.6b. However, as Few pointed out in [26], there is no need to indicate anything if all the data is fine. Alerts should only appear in situations where the icon can provide additional information to the user. Up and down icons (Figure 3.6c) indicate if a particular data point has increased or decreased compared to the past or a specific target. On and off icons serve solely as flags to differentiate between items or data and can have a variety of different icons (Figure 3.6a).

3.6.2 Text

This display includes all non-visual information in the dashboard that is necessary for it to make sense, such as widget names, filters, and other types of information.

3.6.3 Visualization

In order to effectively communicate information through visualizations in a dashboard, it's important to understand the different elements that make up a chart: color, form, spatial position and motion [72]. One of the most important elements is the use of color, which can impact how users perceive the data. Position and form can also play a crucial role in communicating information, as well as the use of motion for animated charts. In this section, we will dive into these elements and explore how they can be used to effectively communicate data in a dashboard.

Color

Color is a crucial element in data visualization and can have a significant impact on the effectiveness of a chart. As stated by Tufte [64], "avoiding catastrophe becomes the first principle in bringing color to information: Above all, do no harm." Color consists of three attributes: hue, saturation, and lightness [26]. Hue refers to the name of the color itself (e.g., blue, yellow, etc.). Saturation represents the purity of the color, with higher saturation indicating a more vivid color. Lightness refers to the brightness of the color, ranging from dark to light.

An effective color design presents information in a way that is easy for the reader to understand the relationship between all the displayed elements. Using the same color for related items is an easy way to label and group them. M. Stone advises in [61] to use between two and three hues and create palettes by varying the saturation of these hues.

To ensure correct color selection, Few in [28] proposed some rules to follow. First, the background color should be consistent and contrast with the chart colors. This means avoiding gradients and selecting colors that are easily visible on top of the background color. Colors should only be used to communicate information and not for styling the chart. Different colors should only be used to display differences in the data. For sequential data, a single hue with varying intensity should be used, while quantitative values should be represented using a dual-ordered palette (a palette with two colors).

However, many color selection tools provide limited control over color value [61], so it is important to have a clear understanding of color in order to select the right colors. In order to facilitate the selection of colors by the user, the application will provide a selection of color ranges instead of individual colors. To start there will be available three sequential ranges: bluescale, greenscale, and redscale these were selected as its easy to differentiate easily low from high numbers and are the three main colors of computing systems. One quantitative range from red to blue. Additionally, a qualitative range formed by random colors. These ranges were selected based on their suitability for different data types and visualization purposes.

Position

Position is a key factor in effectively communicating information through a dashboard. According to Few [26] the top-left and center sections of a dashboard are the areas that provide the greatest emphasis. This is due to reading conventions, which sequence words on a page from left to right and top to bottom, and visual perception, which is drawn to the center of the screen. However, placing information in the center of the dashboard is effective only when it is set apart from its surroundings, such as through the use of white space. Understanding the emphasizing effect of different regions of the dashboard is crucial in making informed decisions about where to place dashboard elements to ensure that they are noticed and effectively convey the desired information to the user, this can be better visualized in figure 3.7.

In user interface design, it is crucial to prioritize the most important elements, which should be positioned at the center and top-left corner of the screen. These areas attract the user's attention and are commonly utilized for primary buttons, critical information, or important features. On the other hand, secondary buttons, labels, or less crucial elements are typically placed in the bottom-right corner. This positioning helps maintain a clear visual hierarchy and assists users in easily locating and interacting with the essential components of the interface.

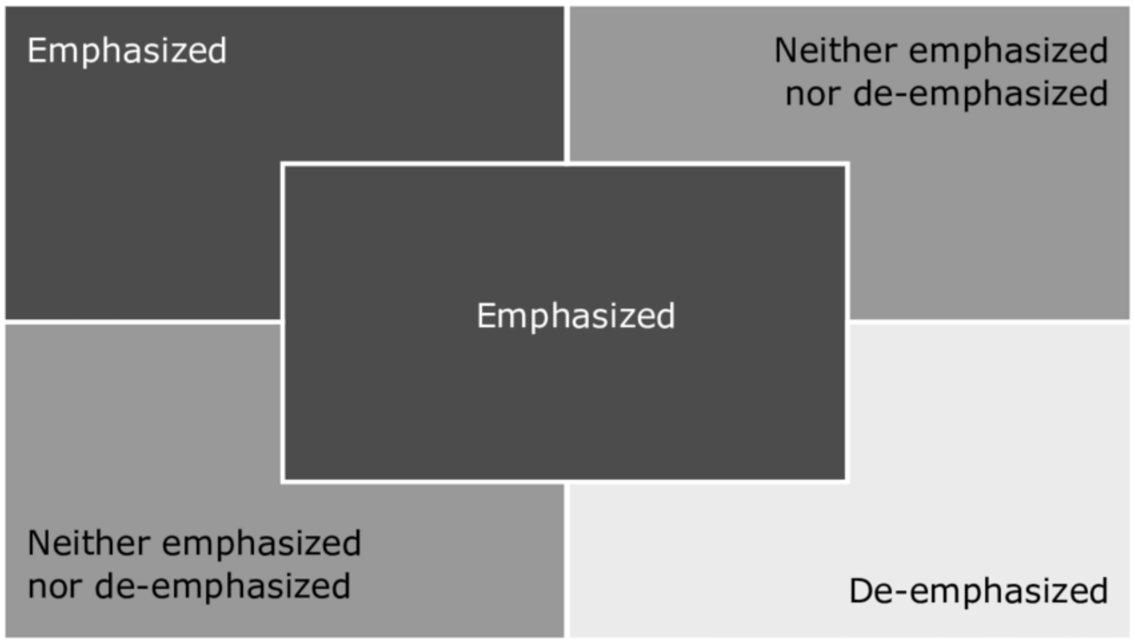

Emphasized
Neither emphasized nor de-emphasized
Emphasized
Neither emphasized nor de-emphasized
De-emphasized

Figure 3.7: Different degrees of visual emphasis based on regions [26]

Motion

Objects in motion, should be used sparingly according to Few [26]. These moving elements can be distracting to users and should only be utilized for real-time data where chart updates or immediate action is required.

3.7 Usability

Usability plays a crucial role in dashboard design as it directly impacts how effectively and efficiently users can achieve their goals and navigate the interface. The evolution of dashboard development now emphasizes the creation of not only functional dashboards but also ones that are user-friendly and effective. Unfortunately, despite its significance, usability is often overlooked in dashboard design [43].

According to Nielsen, usability is assessed based on five key components [46]:

- Learnability: How easy is it for users to accomplish basic tasks during first use?
- Efficiency: Once users have learned the design, how quickly can they perform tasks?
- Memorability: When users return to the design after a period of not using it, how easily can they re-establish proficiency?
- Errors: How many errors are made, how severe are these errors, and how easy is it to recover from the errors?
- Satisfaction: How pleasant is the interface?

The International Organization for Standardization (ISO) defines usability as the extent to which a product can be used by specified users to achieve specific goals with effectiveness, efficiency, and satisfaction in a given context of use [4]. This definition not only takes into account the user and their context but also emphasizes the importance of achieving the desired goals. However, assessing user goals and their fulfillment poses empirical challenges.

According to C. Burnay et al. [14], effective dashboards should enable users to achieve their goals quickly and effortlessly by maintaining an organized and structured presentation of information within a contextual framework. In addition, dashboards should offer a clear content organization that allows users to navigate with ease and a flexible interface that caters to individual user preferences and control [15]. It is also crucial to provide users with adequate support through features such as a help function, error avoidance, and recovery mechanisms [15]. By incorporating these elements, users can easily understand the dashboard's functionality and recover from errors if they occur. Furthermore, a well-designed dashboard enhances memorability, allowing users to quickly recall how to perform tasks during subsequent visits [15].

3.7.1 Ease of Use

In "Usable Usabilty Simple Steps for Making Stuff Better" by Eric Reiss [52] defines ease of use as the physical properties of usability "It does what I want it to do". Reiss further categorizes ease of use into five key aspects:

- Functional - Users expect the product to work as intended, with buttons and links performing the expected actions;
- Responsive - Users should receive feedback that confirms the system is working and indicates its current state;
- Ergonomic - The interface should be designed in a way that allows users to easily interact with it, including clear visibility, intuitive clicking, poking, twisting, and turning;
- Convenient - The product should provide users with easy access to the features and information they need, with everything conveniently located and organized;
- Foolproof - Designers should anticipate potential mistakes or errors users may make and implement safeguards or guidance to prevent or mitigate them.

Functionality

Reiss emphasizes three essential keys to ensuring functionality in a product. First, buttons and links should reliably perform their intended actions when clicked or activated. Second, the navigation system should be responsive and enable users to move through the product smoothly. Lastly, the speed of the product's response should meet user expectations, providing acceptable performance.

A crucial aspect of ease of use is the simplicity and intuitiveness of navigation. Users should be able to navigate through the product effortlessly, quickly finding the information or features they are seeking. Eric Reiss highlights the significance of clear labeling for navigation elements, utilizing familiar terms, and maintaining consistent placement across the site or application.

Responsive

A responsive user interface employs various techniques to enhance user interaction. One such technique is the use of transitional effects, where visual responses occur in direct correlation to user actions. For example, when a user hovers over an interactive button, the cursor changes to a hand icon, or when a link changes color upon being highlighted. These transitional effects provide immediate feedback to the user, reinforcing the connection between their actions and the system's response.

Another essential aspect of a responsive UI is the implementation of responsive mechanisms, which serve as "receipts" from the interface to the user. These mechanisms inform users that their actions have triggered a specific outcome or process. For instance, displaying a popup dialog that dims the rest of the screen creates a clear visual indication of a response, or zooming in or out during a specific operation provides a tangible feedback mechanism.

Conversely, a poorly designed responsive UI can lead to feelings of Fear, Uncertainty, Doubt (FUD) among users. When users lack confidence in their actions and fear breaking the system or making unintended mistakes, they become hesitant and uncertain about using the application. This uncertainty arises when users believe that any choice they make may result in negative consequences.

The key takeaway is that, similar to a conversation, incorporating response mechanisms that provide sensory feedback can help alleviate FUD. By ensuring that users receive clear and immediate feedback for their actions, designers can enhance user confidence, streamline interactions, and eliminate unnecessary hesitation.

Ergonomic

Ergonomics encompasses the study of designing devices and systems that align with the physical and psychological capabilities of users. It aims to optimize user experience, comfort, and efficiency by considering human abilities and limitations. When designing interfaces, it is essential to regard the cursor as an extension of the user's fingers. Similar to the fingers, there are certain actions that the user can perform effectively, and others that may pose challenges.

One crucial principle in ergonomics is Fitts's Law [52], which provides insights into the relationship between movement time, target distance, and target size. According to Fitts's Law, the time required to move rapidly to a target area depends on two factors: the distance to the target and the size of the target.

Convenient

When venturing into unfamiliar domains or exploring new territory, individuals often gravitate towards their comfort zones. This inclination arises from our innate desire for familiarity and the reassurance that comes with established routines and patterns. However, it is crucial to recognize the importance of seeking input from diverse sources to avoid the pitfall of solely addressing our own needs while overlooking the needs of other users.

One effective approach to broaden our understanding and consider different perspectives is through the use of personas. Personas allow us to empathize with and gain a deeper understanding of the motivations, goals, and behaviors of various user types. This enables us to design experiences that cater to the diverse needs of our target audience, leading to more inclusive and user-centered solutions.

Foolproof

Creating a foolproof interface requires implementing certain strategies to guide and assist users throughout their interactions. One key aspect is to provide clear reminders when something may be missing or incomplete, ensuring that users are alerted to any necessary actions before proceeding. By eliminating unavailable options and providing informative error messages, the interface enhances usability and empowers users to take the correct course of action promptly.

Alerts serve as effective means of notifying users about errors, changes in system state, or any other important information that requires their attention. However, it is essential to strike a balance between alerting users without overwhelming them or triggering a "boy who cried wolf" syndrome, where users start ignoring important alerts due to excessive or irrelevant notifications.

To prevent users from making mistakes, certain techniques can be employed, such as removing or disabling options that are not applicable or currently unavailable. For instance, graying out disabled options or hiding them from view can be used. However, it is important to provide proper context and reasoning for the disabled or hidden options to avoid confusion. Users need to understand why certain options are not accessible or have disappeared, otherwise, they may continue attempting to access them without realizing the conditions for their availability.

Additionally, users often expect similar abilities and functionalities to transcend across different environments or applications. Therefore, when building applications with similar functionality, it should be avoided unnecessarily deviating from established conventions and creating unfamiliar or unusual ways to accomplish the same tasks. Consistency in design and interaction patterns helps users leverage their existing knowledge and reduces the learning curve, leading to a more intuitive and user-friendly experience.

3.7.2 Elegance and Clarity

On the same book E. Reiss [52] highlights the psychological properties of usability, namely elegance and clarity, which encompass users' expectations of how a system should behave. He further categorizes these properties into five key aspects:

- Visible - Users should be able to perceive and visually understand the elements and information presented to them;
- Understandable - Users should have a clear comprehension of what they are looking at and how the system functions;
- Logical - The elements and procedures within the system should make logical sense, ensuring a coherent and rational user experience;
- Consistent - Users expect the system to adhere to consistent rules and patterns, avoiding unexpected changes or surprises in its behavior;
- Predictable - Users should have a clear sense of what will happen next when they perform an action, allowing them to anticipate the system's response.

Visible

Upon entering a page, users expect to find the most important functions prominently displayed, right where they can easily locate them. To ensure optimal usability, it is crucial to eliminate any irrelevant elements that may distract users from the essential features. In essence, the visibility of an item depends on several factors: its placement within the user's focus area, the absence of obstructions, its recognizability, and its existence in the interface.

Understandable

This category encompasses both text and icons within the interface. In terms of text, it is essential to ensure clarity and conciseness, as users may interpret information differently based on their backgrounds and perspectives. Clear and straightforward language helps minimize confusion and facilitates comprehension for a diverse user base.

When it comes to icons, using familiar and recognizable icons from established platforms like Microsoft, Apple, or Google is recommended. Leveraging commonly used icons enhances user experience by leveraging existing knowledge and familiarity across applications. By utilizing familiar icons, users can easily associate them with their respective functions, promoting ease of use and reducing the learning curve.

Logical

The logical category in usability involves different forms of reasoning to facilitate understanding and decision-making. Deductive reasoning follows a logical sequence, where if certain statements are true, then their logical implications must also be true. Inductive reasoning, on the other hand, relies on past observations and suggests the probability of something being true based on those observations. It involves making judgments based on evidence but does not guarantee absolute truth.

Retroductive inference involves learning from one situation and applying that knowledge to similar but new situations. It is akin to understanding how to navigate an unfamiliar airport based on previous experiences with airports. This form of inference allows users to transfer their knowledge and skills to new contexts.

These forms of reasoning play a role in creating usable interfaces by guiding the design of logical and intuitive workflows, ensuring that users can make informed decisions and predict the outcome of their actions.

Consistent

In the context of interfaces, consistency entails using the same words, labels, and design elements consistently throughout the system. It ensures that users can rely on their prior knowledge and experiences to understand and interact with the interface effectively.

Consistency is crucial in button labels, where using different words or phrases to convey the same function can lead to confusion. For example, if "Submit" is consistently used as a button label, suddenly changing it to "Send" or "Accept" can create uncertainty for users. Similarly, consistency should be maintained in signage within public facilities to provide standardized information.

Predictable

Predictability is a key aspect of usability that focuses on ensuring that interfaces and interactions behave in a consistent and expected manner. When something is predictable, it performs as users anticipate, reducing confusion and enhancing user experience.

Informing users about what to expect before they reach a certain point or perform an action helps set their expectations and reduces surprises. Clearly communicating expectations to users regarding their role or responsibilities in a process fosters predictability and enables smoother interactions.

Providing information about the number of steps in a multi-step process enhances predictability by giving users an understanding of the overall workflow and allowing them to plan accordingly. Ensuring that users understand the desired outcome of the process they are currently engaged in further contributes to predictability.

Placing elements and features in locations where users naturally expect to find them aligns with their mental models, making interactions more intuitive and predictable. Additionally, incorporating visible signals that alert users to invisible conditions or potential issues helps them anticipate and respond appropriately.

3.8 Charts

Charts are the key component of every dashboard. In this section, the various types of charts available, their uses, and common errors and pitfalls to avoid will be presented. It will also be discussed the importance of choosing the right type of chart for the data being presented. By understanding the basics of charts and the role they play in a dashboard, we can ensure that the charts available in the platform as well as their features are the best for the users to create their dashboards.

3.8.1 Types of Charts

For the initial version of the application, the AlticeLabs team has chosen a set of charts, which includes line, scatter, pie, vertical and horizontal bar, stacked, and a column-line chart. Extensive research has been conducted on these charts to offer users the most effective tools for creating and managing them. The research draws upon the valuable resources of the Data Viz project [25], The Data Visualization Catalogue [53], and the Financial Times visual vocabulary guide [63].

Line Chart

The line chart is the standard method for visualizing a changing time series or continuous series. It allows for easy interpretation of slopes, with upward slopes indicating increases and downward slopes indicating decreases in values. Line charts are effective in identifying patterns that reveal trends. They can be used in conjunction with other data series to facilitate comparisons, but it is important to limit the number of visible series to avoid cluttering the chart. It is generally recommended to display no more than five data series simultaneously.

Scatter Plot

A scatter plot is a commonly used method to visualize the relationship or correlation between two variables, each represented on its respective axis. It allows for the identification of positive patterns (when both variables increase together), negative patterns (when one variable increases as the other decreases), or null patterns (no relationship). To aid analysis, lines or curves can be added to the chart, such as a best fit or trend line, which enables estimation through interpolation.

Bar Chart

The most common way to present discrete and numerical comparisons across different categories, particularly qualitative ones, is through bar charts. The bars can be arranged in any order, but it is advisable to order them logically to facilitate comparison. Bar charts can be displayed either vertically or horizontally, with the orientation determining the placement of the axis. Horizontal bars have the advantage of accommodating more readable labels, making them suitable for larger datasets or narrow layouts. It is important to ensure that the axis always starts at zero to maintain accurate representation.

Pie Chart

The pie chart is the easiest chart to read and provides an ideal representation of the proportional distribution of data in a part-to-whole relationship. However, it also has several limitations. It is recommended to have no more than six to eight categories in a pie chart because as the number of categories increases, the size of each segment becomes smaller and it becomes more challenging for the reader to accurately compare the sizes. The reader's ability to distinguish the sizes via area becomes less accurate when compared to via length, like the bar chart does.

Stacked Chart

Stacked charts are particularly effective in visually representing both the size and proportion of data simultaneously. In a stacked chart, multiple datasets are stacked on top of each other, enabling comparisons among the smaller subcategories that form the larger category. The total value of the bar is obtained by summing all the segment values. This type of chart is ideal for comparing the overall amounts across each segmented bar. However, it is important to note that as more segments are added, the stacked chart can become increasingly challenging to read and comprehend.

Column-line Chart

This chart includes both bars and line chart, each with a correspondent y axis. Showcases the relationship between an amount (columns) and a rate (line).

3.8.2 Integrity of Charts

A chart is considered to be distorted if it fails to accurately reflect the numerical representation it is supposed to depict [65]. E. Tufte developed a metric to measure this distortion, known as the Lie Factor [65]. The Lie Factor is calculated as the ratio between the size of the effect displayed in the graph and the actual size of the effect. A larger Lie Factor indicates a higher degree of distortion. As seen in Figure 3.8, a chart with a high Lie Factor can be misleading, as the increase in size of each line does not accurately reflect the corresponding increase in value. In contrast, Figure 3.9 shows a well-designed chart that accurately and effectively conveys the relevant information without distorting the data.

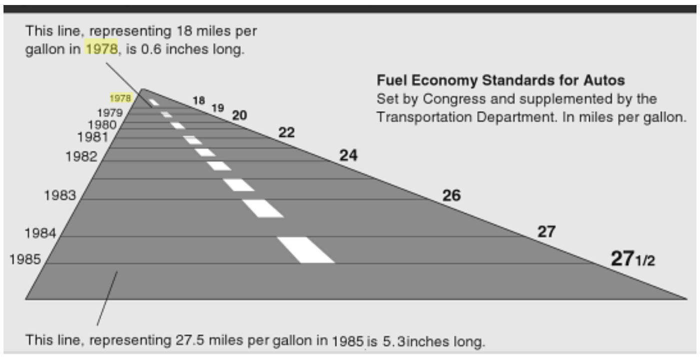


Figure 3.8: Distorted Chart representing the fuel economy standards per year [65]

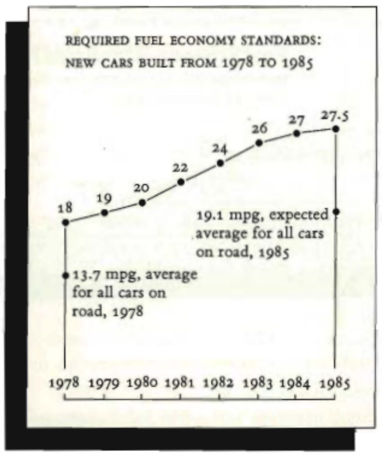


Figure 3.9: Accurate chart representing the fuel economy standards per year [65], sourced from a print of the original book due to unavailability of a higher quality image.

Another common mistake arises from human's tendency to underestimate differences in 2-D areas, and therefore it is important to be mindful when using 2-D areas of varying sizes to encode quantitative values, especially in a dashboard where quick interpretation is essential [26]. This mistake has been referred to as the "Shrinking Doctor" effect by Tufte and can be seen in Figure 3.10. The example demonstrates how the smaller images appear much smaller in size than the actual decrease in percentage, making it difficult for the average viewer to detect the true reduction.

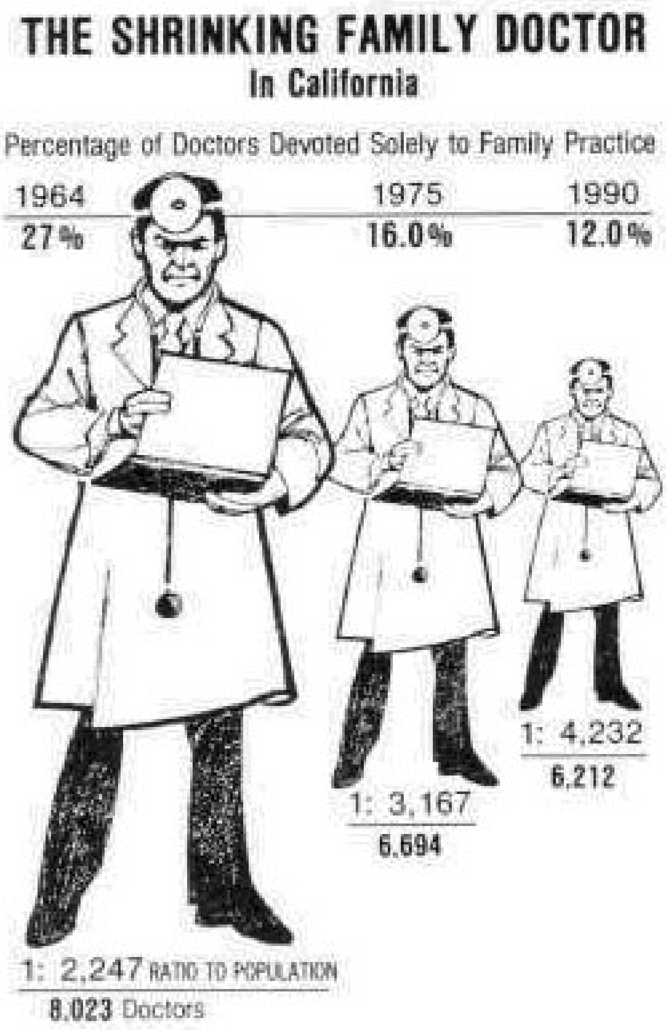


Figure 3.10: Shrinking doctor effect [65], sourced from a print of the original book due to unavailability of a higher quality image.

Dashboards should also focus on showcasing variations in data, not variations in design. Figure 3.11 provides an example of how an irregular scale (with the last metric only covering 4 years instead of 10) can be utilized to deceive viewers into perceiving a decline that is actually smaller than depicted.

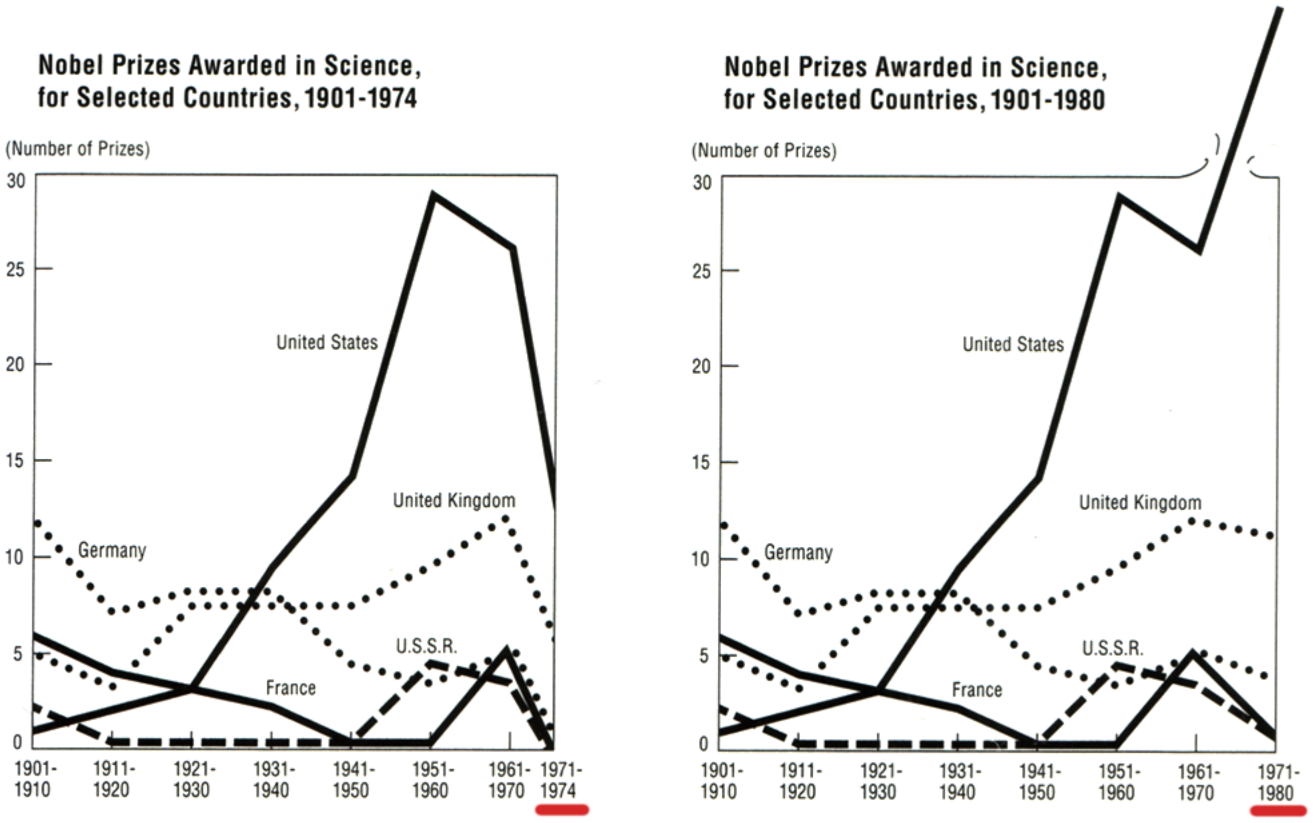


Figure 3.11: Example of design variation [65]

Lastly, it's important to provide context when presenting data, as data without context is meaningless and can often lead to misinterpretation or misquotation. If a chart doesn't answer the question "Compared to what?" then it lacks context and needs to be framed accordingly.

3.8.3 Automatic Visualization Recommendations

The design of charts depends on the user's skill and the context of the data to be visualized [38]. Since not all users of the platform will have the necessary skills, one way to simplify the process is by automatically selecting an appropriate chart or offering a set of chart options that best represent the data. Although this topic is not the main focus of the current work, it will be briefly discussed and should be better explored in future works.

In the work of P. Kaur and M. Owonibi [38], a visual mapping method through rules is introduced. This method analyzes the data and applies a set of rules to select the correct visual mark, scale, and channel. This method is used in Tableau's Show Me [42], Voyager [74], and Manyeys [69].

VizDeck [39] proposes a method that only considers the input data and learns from the user's choices over time. However, this method requires the user to have some experience as if the users makes wrong choices the system will also suggest them wrong choices.

VISO [70] is a context-aware and knowledge-assisted approach for the recommendation of visualization components. It has the capability to annotate both data sources and visualization components. Using VISO, M. Voigt et al. [71] added a layer of rankings and rules to provide recommendations based on the suitability for the current context of the dashboard.

Recently, the use of neural networks has become popular in the field of visualization recommendations. Systems like VizML [34] and Data2Vis [67] are training their models using information from knowledgeable communities, like the Plotly community, to learn how different visualizations were created for different data sets. By using this information, they can generate outputs that can even include the correct axis placement for each variable.

3.9 Key Performance Indicators

The ABC records and stores various parameters and metrics. This section aims to provide a brief overview of the specific data and fields that will be available to chart on the first version of the platform. The descriptions of all these fields are taken directly from the "Service Detail Record (SDR) Table Specification," which is incomplete, and because of that, some of the fields cannot be fully explained.

The ABC's KPIs are organized into six distinct database views: Communications or Calls and SMS, Hunt Group, Waiting Queues, Pre-Answer, IVR Menu, and End-Causes. All of the parameters are recorded in a single table named "SDRs" This data is then transformed and loaded into eighteen different views. These views correspond to the six groups of parameters, each of which is further divided into three subgroups based on the time intervals of the data collected (15 minutes, hourly, or daily). Each of these subgroups contains different fields that provide valuable insights into the data collected. For the first version of the platform, the decision was made as a team to focus on the data from the Waiting Queues and Communications KPIs, as they were deemed the most critical of the group.

Even though most of the fields of each group view are different, some are shared between all. These are the date of record, the customer ID, the Direct Dial-In (DDI) that identifies the origin of the call and the number of events.

3.9.1 Waiting Queues

As the name suggests, the Waiting Queues KPI tracks all the statistics related to the customers who are waiting for their calls to be answered by an agent.

This KPI tracks how long the customer waited on the queue and the call duration, as well as the minimum, maximum, and average of every waiting queue and call. It also provides information on the party that ended the call and why, an indication of whether the service is active or not, if the call was answered or not, and if there was a forwarding of calls or not (which can lead to a further waiting queue). The ID of the waiting queue is also registered.

3.9.2 Communications

This KPI saves all the metrics of the user during communication with an agent.

It stores the agent identifier, the AS cluster node, the ID of the access used and terminal used by the agent, the type of call, location, and transaction made by the customer, as well as the ring times and call duration and whether or not the customer's call was answered.

3.10 Business Intelligence Tools

Now that dashboards, the types of dashboards, and how to design a dashboard are defined, this section will examine some tools created by major players in the BI industry for creating and managing dashboards. It will also assess the features and design of these tools. Most BI tools nowadays offer similar features, with differences mainly in data connection types, prices, and key focus points.

3.10.1 PowerBI

Microsoft PowerBI is a widely used BI tool for end-to-end analytics, providing users with multiple ways to connect and analyze their data. PowerBI focuses on:

- Quick information breakdown: It facilitates access to data exploration, helping users make quick informed decisions;
- Refreshing data: Not all data sources support live querying, so PowerBI offers automatic refresh for dashboards, which can be set for a specific time of day;
- Cross-platform: It supports embedded web applications, Android, and iOS;
- Ease of use: PowerBI has a low learning curve and good UX.

However, chart customization is largely limited, which can be frustrating if a specific requirement cannot be met. Figure 3.12 shows a screenshot of the creation of a widget on PowerBI.

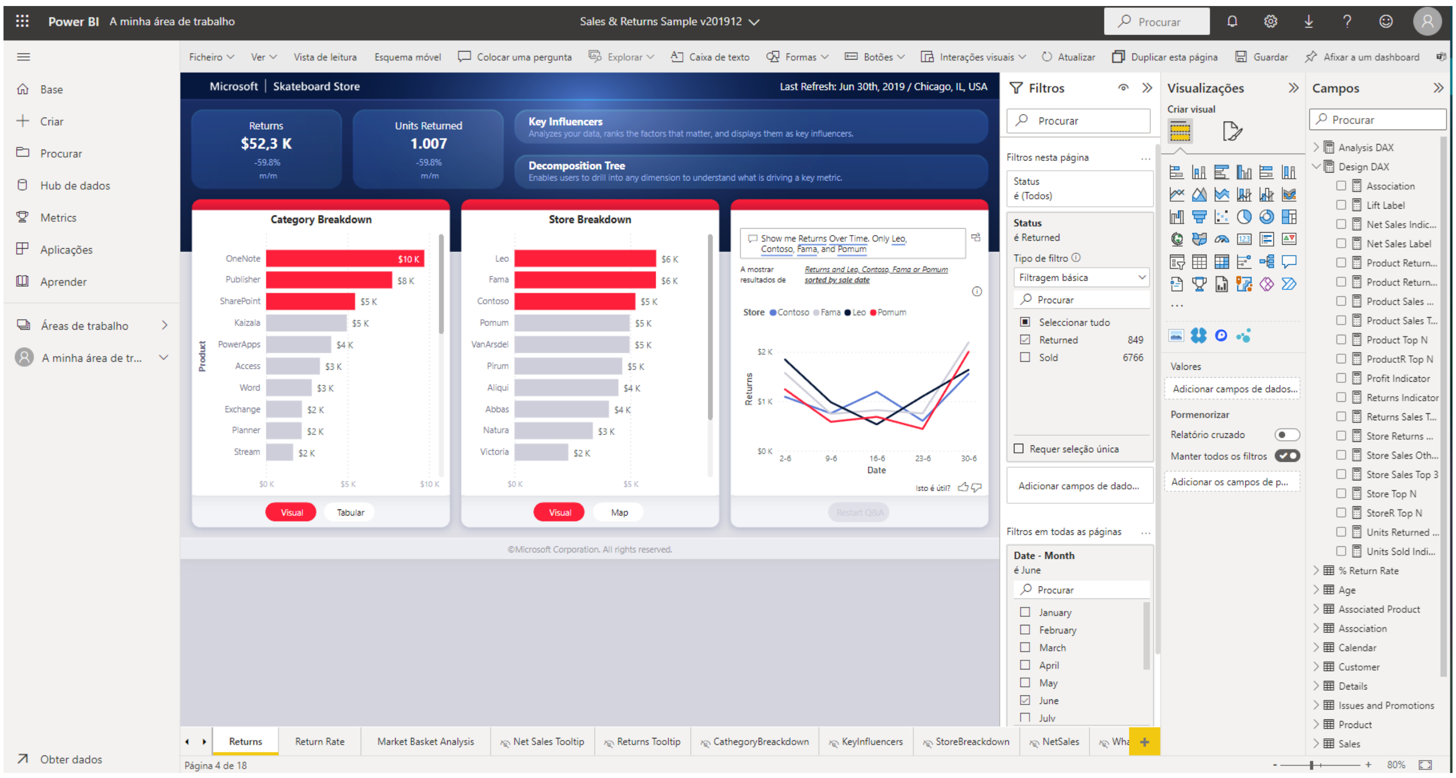


Figure 3.12: Example of creation of a new dashboard in PowerBI. Image taken from a dashboard created by me in PowerBI.

3.10.2 Tableau

Tableau distinguishes itself from other BI tools with its unique aesthetics, which are powered by Pixar artists, while still offering reliable performance. It stands out for the user experience it offers to users, being one of the easiest platforms to use for BI.

- Visualisation capabilities: Tableau converts most data into interactive, comprehensive, and functional results in every possible way;
- High performance: Tableau is unique for its robust and reliable performance, being fast even for large datasets;
- Cross-platform: Supports both android and IOS.

Figure 3.13 shows the drag and drop mechanism used by tableau to create a new chart.

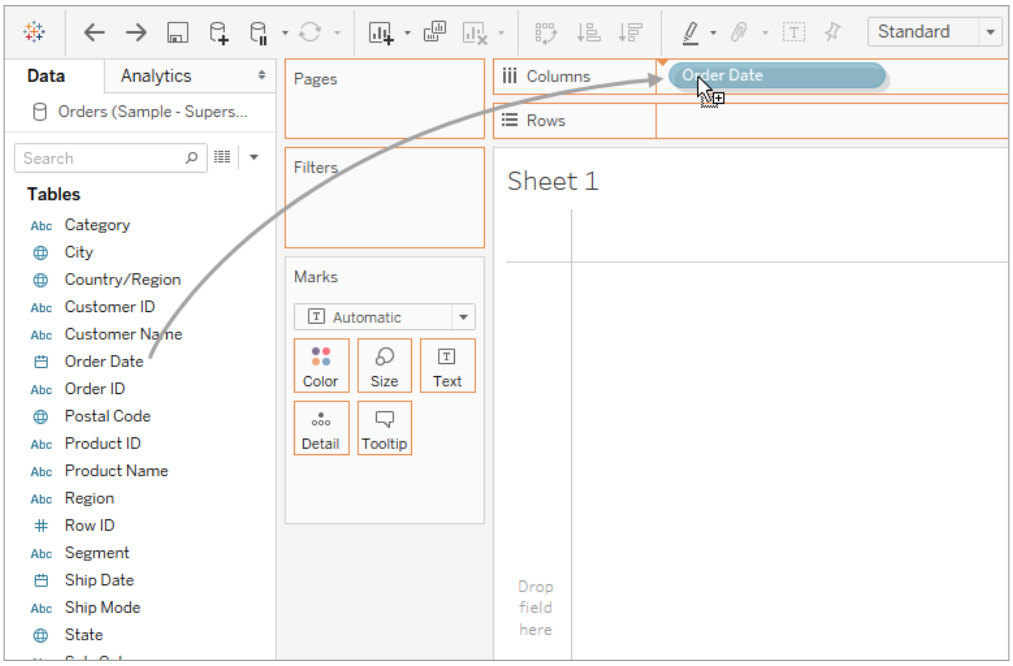


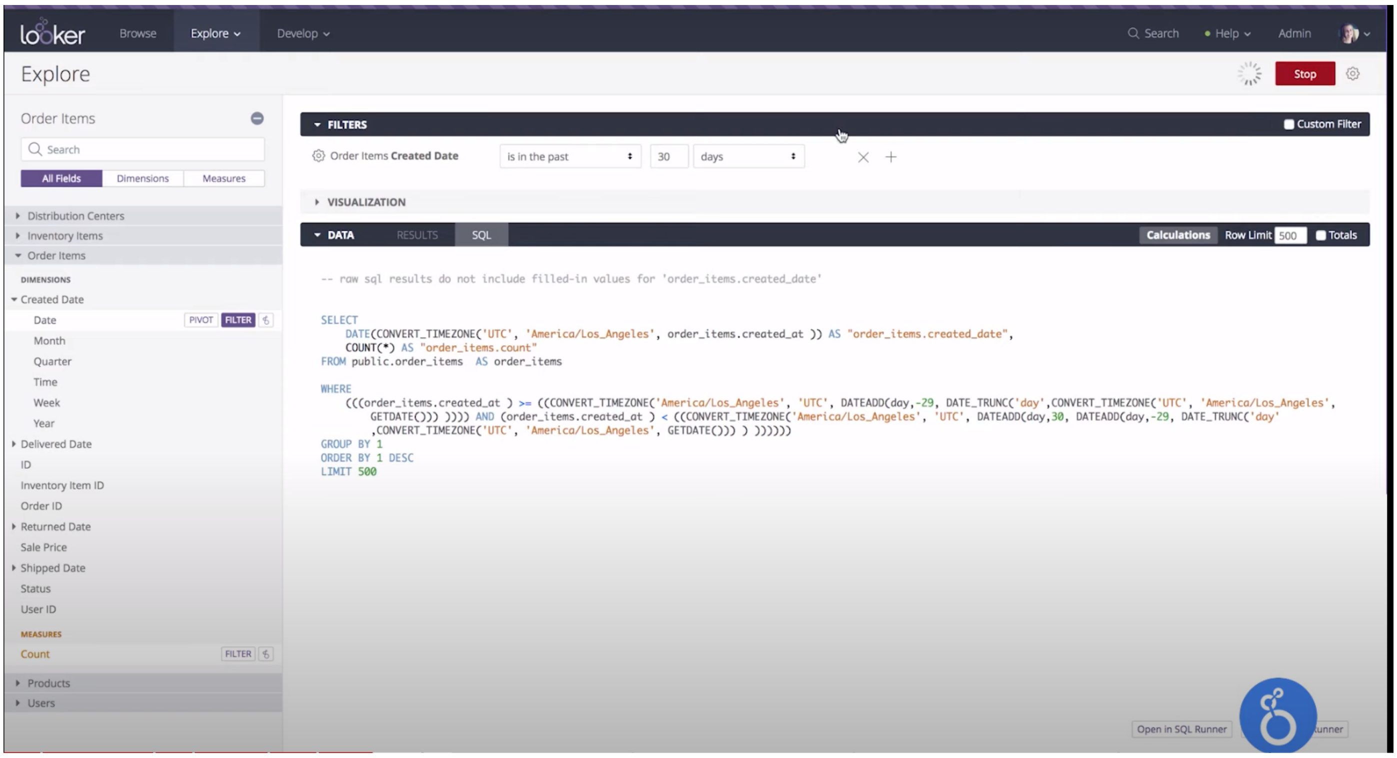
Figure 3.13: Example of creating a new chart in Tableau. Image taken from a Tableau tutorial [62]

3.10.3 Looker

Looker is Google's new product for the BI market. It goes beyond BI and integrates seamlessly into business workflows, embedding into many third-party systems and enabling companies to build their own data applications.

- Collaboration: Offers layers where analysts can define business logic and communicate with other users;
- Cloud-based: As a newer platform takes advantage of the cloud scalability and performance because this doesn't rely on stale data, allowing for real real-time data charts;

However, Looker has a steep learning curve for users, as many of its features are new for this type of tool. The figure 3.14 shows an example of creation of a chart using the database query mode on looker.



```
-- raw sql results do not include filled-in values for 'order_items.created_date'


SELECT
    DATE(CONVERT_TIMEZONE('UTC', 'America/Los_Angeles', order_items.created_at )) AS "order_items.created_date",
    COUNT(*) AS "order_items.count"
FROM public.order_items  AS order_items

WHERE
    (((order_items.created_at ) >= ((CONVERT_TIMEZONE('America/Los_Angeles', 'UTC', DATEADD(day,-29, DATE_TRUNC('day',CONVERT_TIMEZONE('UTC', 'America/Los_Angeles',
        GETDATE())) )))) AND (order_items.created_at ) < ((CONVERT_TIMEZONE('America/Los_Angeles', 'UTC', DATEADD(day,30, DATEADD(day,-29, DATE_TRUNC('day'
        ,CONVERT_TIMEZONE('UTC', 'America/Los_Angeles', GETDATE())) ) ))))))
GROUP BY 1
ORDER BY 1 DESC
LIMIT 500
```

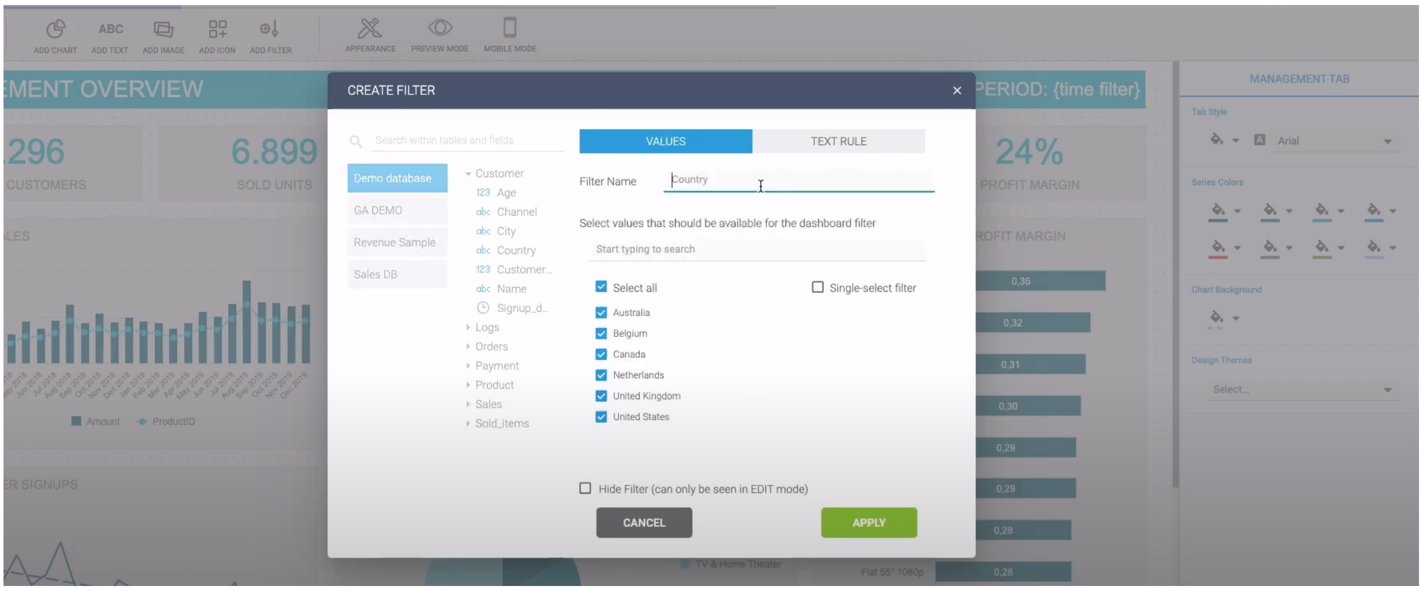
Figure 3.14: Example of loading data to create a new chart in Looker. Image taken from a video by Looker [41]

3.10.4 Datapine

Datapine allows for the creation of powerful dashboards with customized dashboards and alerts. It also allows users to create custom metrics using its query tool. It has a low learning curve, with most tasks involving drag-and-drop, making it easy to create dashboards.

- Easy predictive analytics: It includes a predictive system based on a forecast engine. The user only needs to select an indicator, data points, and model quality;
- Report options: It allows for embedding and emailing reports;
- Alarms: It offers multiple options for alarms, including growth, drivers, or even conditional scenarios.

Figure 3.15 shows how to apply filters on dashboard created by datapine.



Filter Name: Country

Select values that should be available for the dashboard filter

- [✓] Select all
- [ ] Single-select filter
- [✓] Australia
- [✓] Belgium
- [✓] Canada
- [✓] Netherlands
- [✓] United Kingdom
- [✓] United States
- [ ] Hide Filter (can only be seen in EDIT mode)

Figure 3.15: Creating a new filter in Datapine. Image taken from a Datapine video [21]

3.11 Analytics in Unified Communications as a Service Tools

Big data analytics for unified communications is becoming increasingly important for entrepreneurs, with more and more billions of dollars being invested in software development every year [1]. Unified Communications as a Service (UCaaS) is a cloud-based communication and collaboration solutions that integrates all of UC benefits into a single platform. Analytics in UCaaS empowers enterprises to gain valuable insights into their communication patterns, usage trends, and user behaviors. In the following section it will be explored how UCaaS implement their analytic platforms in order to get a better insight on what features to add to ABC analytics system. Please note that as these tools are behind paywalls, not every detail and functionality of the system was able to be documented.

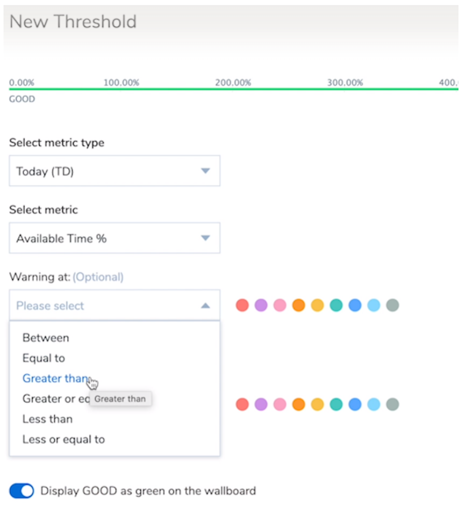
3.11.1 8x8

8x8 is the largest UCaaS platform today, listed multiple times as a leader in Gartner's Magic Quadrant for UCaaS [45].

According to 8x8, their analytics offer "highly-visual and intuitive dashboards, giving an instant understanding of your contact center performance" [2]. 8x8 analytics focuses more on providing users with pre-defined dashboards with all the data they need, while still providing the necessary tools to create a custom one. Their key features are:

- Creation of custom widgets;
- Pre-defined dashboards;
- Historical data allowing the creation of visual data charts with historical information;
- Real-time data providing the visualization of indicators for queues and agents in short time intervals;
- Email reports with snapshots of the dashboard;
- Custom Metrics.

Some of these features can be found in figures 3.16 and 3.17 and a dashboard page is showed in figure 3.18.


New Threshold

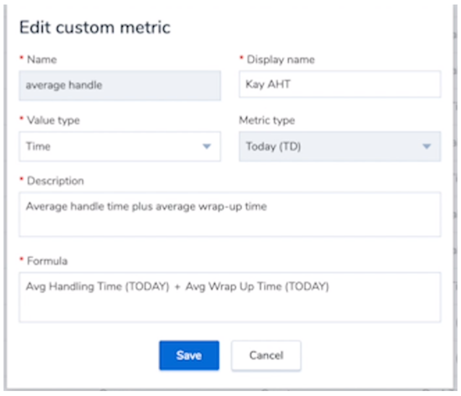
Select metric type: Today (TD)

Select metric: Available Time %

Warning at: (Optional) Please select

Display GOOD as green on the wallboard

Figure 3.16: Creating alerts in 8x8 dashboard. Image taken from [3]


Edit custom metric

* Name: average handle

* Display name: Kay AHT

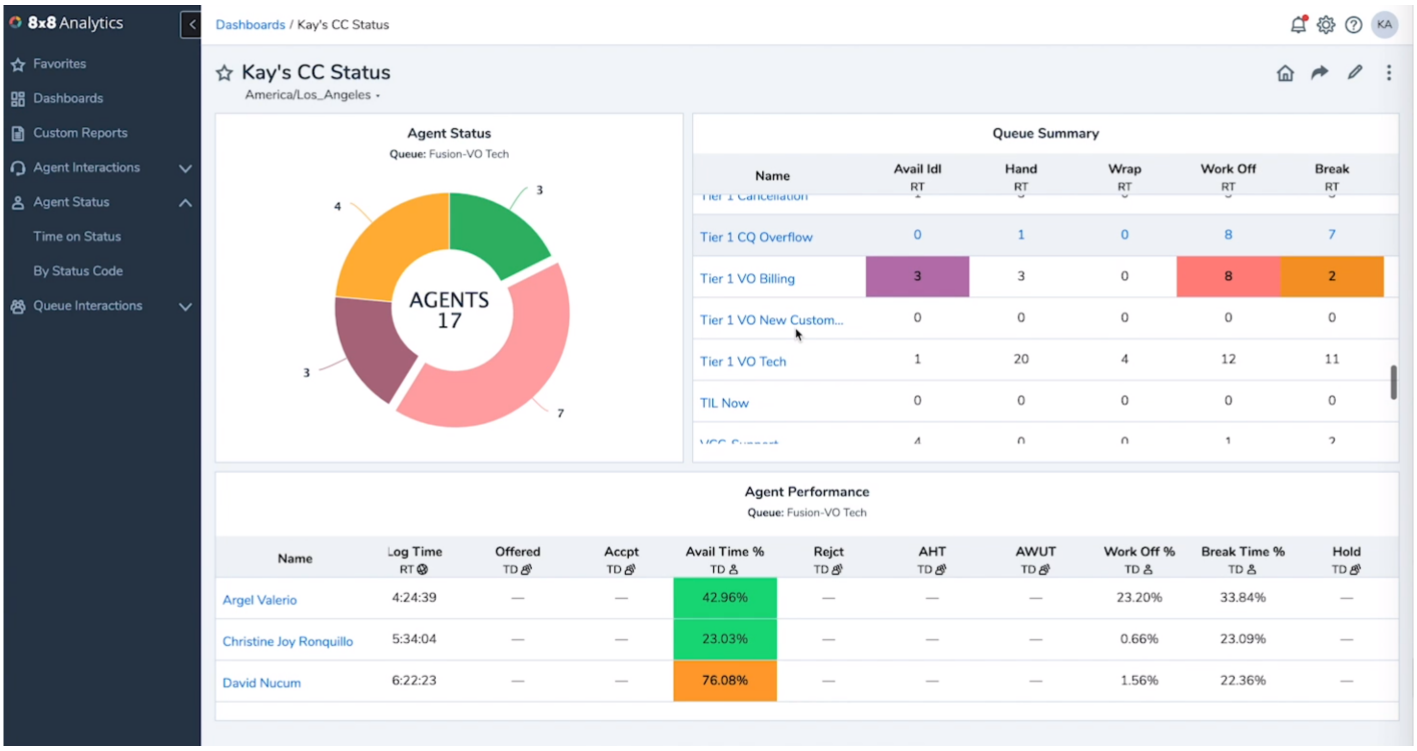
* Value type: Time

Metric type: Today (TD)

* Description: Average handle time plus average wrap-up time

* Formula: Avg Handling Time (TODAY) + Avg Wrap Up Time (TODAY)

Figure 3.17: Creating a custom metric. Image taken from [3]



Queue Summary

Name	Avail Idl RT	Hand RT	Wrap RT	Work Off RT	Break RT
Tier 1 CQ Overflow	0	1	0	8	7
Tier 1 VO Billing	3	3	0	8	2
Tier 1 VO New Custom...	0	0	0	0	0
Tier 1 VO Tech	1	20	4	12	11
TIL Now	0	0	0	0	0

Agent Performance
Queue: Fusion-VO Tech

Name	Log Time RT	Offered TD	Accpt TD	Avail Time % TD	Rejct TD	AHT TD	AWUT TD	Work Off % TD	Break Time % TD	Hold TD
Argel Valerio	4:24:39	—	—	42.96%	—	—	—	23.20%	33.84%	—
Christine Joy Ronquillo	5:34:04	—	—	23.03%	—	—	—	0.66%	23.09%	—
David Nucum	6:22:23	—	—	76.08%	—	—	—	1.56%	22.36%	—

Figure 3.18: Example of a dashboard page in 8x8. Image taken from [3]

As for KPIs 8x8 has a vast list of available KPIs for their charts like interactions by agents and status about queues and agents.

3.11.2 Ring Central

RingCentral is another major player in UCaaS, competing with 8x8 every year for the top spot. RingCentral uses call center metrics to understand workforce or customer behavior patterns, allowing businesses to make the most effective use of their business phone system. RingCentral's key focus points are:

- A variety of KPIs available for agents, calls, web calls, and even rooms, which allows enterprises to focus more on strategy;
- The variety of KPIs allows for a more complex and defined report system;
- Monitoring of device health, allowing their clients to know when any device starts to malfunction.

Although RingCentral offers a wide range of key performance indicators (KPIs), their dashboard customization options are somewhat limited. Users are restricted to three types of widgets: KPI number, table, and trend. Figure 3.19 demonstrates the process of creating a widget, while figure 3.20 showcases an example of a table widget. Furthermore, figure 3.21 displays a RingCentral dashboard.

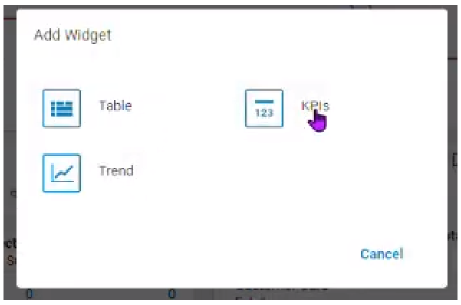
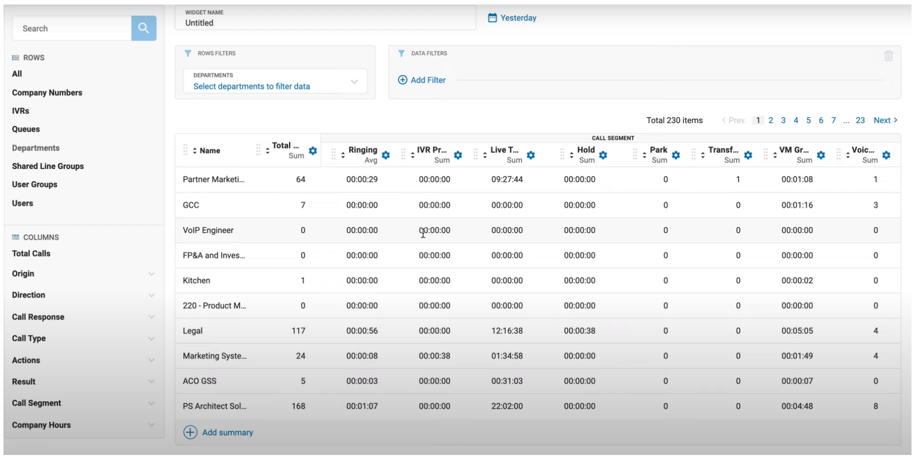


Figure 3.19: Creating new widget in RingCentral. Image taken from [66]



WIDGET NAME: Untitled

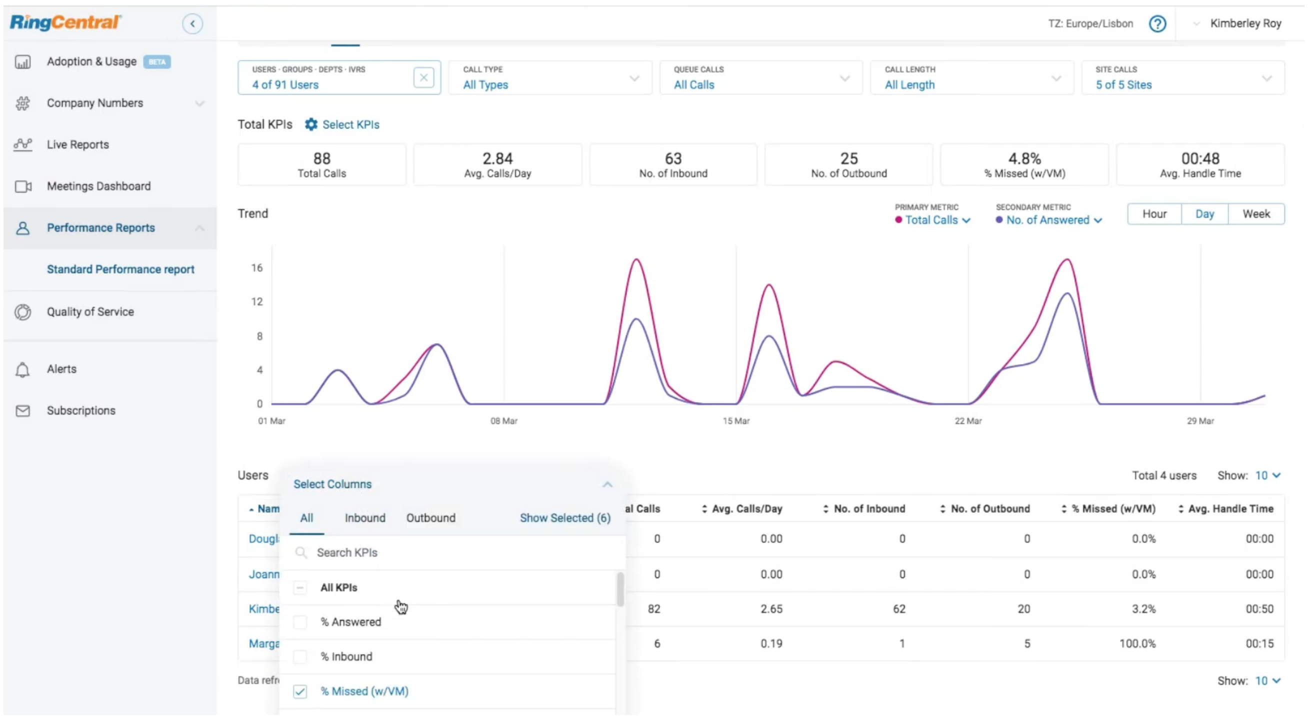
DEPARTMENTS: Select departments to filter data

Total 230 items ‹ Prev. 1 2 3 4 5 6 7 ... 23 Next ›

		CALL SEGMENT							
Name	Total ... Sum	Ringing Avg	IVR Pr... Sum	Live T... Sum	Hold Sum	Park Sum	Transf... Sum	VM Gr... Sum	Voic... Sum
Partner Marketi...	64	00:00:29	00:00:00	09:27:44	00:00:00	0	1	00:01:08	1
GCC	7	00:00:00	00:00:00	00:00:00	00:00:00	0	0	00:01:16	3
VoIP Engineer	0	00:00:00	00:00:00	00:00:00	00:00:00	0	0	00:00:00	0
FP&A and Inves...	0	00:00:00	00:00:00	00:00:00	00:00:00	0	0	00:00:00	0
Kitchen	1	00:00:00	00:00:00	00:00:00	00:00:00	0	0	00:00:02	0
220 - Product M...	0	00:00:00	00:00:00	00:00:00	00:00:00	0	0	00:00:00	0
Legal	117	00:00:56	00:00:00	12:16:38	00:00:38	0	0	00:05:05	4
Marketing Syste...	24	00:00:08	00:00:38	01:34:58	00:00:00	0	0	00:01:49	4
ACO GSS	5	00:00:03	00:00:00	00:31:03	00:00:00	0	0	00:00:07	0
PS Architect Sol...	168	00:01:07	00:00:00	22:02:00	00:00:00	0	0	00:04:48	8

Figure 3.19: Creating new widget in RingCentral. Image taken from [66]

Figure 3.20: Creating table widget in RingCentral. Image taken from [66]



USERS · GROUPS · DEPTS · IVRS: 4 of 91 Users | CALL TYPE: All Types | QUEUE CALLS: All Calls | CALL LENGTH: All Length | SITE CALLS: 5 of 5 Sites

Total KPIs Select KPIs

88	2.84	63	25	4.8%	00:48
Total Calls	Avg. Calls/Day	No. of Inbound	No. of Outbound	% Missed (w/VM)	Avg. Handle Time

Users — Total 4 users Show: 10

Nam	al Calls	Avg. Calls/Day	No. of Inbound	No. of Outbound	% Missed (w/VM)	Avg. Handle Time
Dougl	0	0.00	0	0	0.0%	00:00
Joann	0	0.00	0	0	0.0%	00:00
Kimbe	82	2.65	62	20	3.2%	00:50
Marga	6	0.19	1	5	100.0%	00:15

Show: 10

Figure 3.21: Example of a performance dashboard page in RingCentral. Image taken from [66]

3.11.3 Dialpad

Dialpad is a newer platform that is quickly rising in Gartner's Magic Quadrant for UCaaS [45]. Dialpad analytics is more focused on contact center KPIs, gathering data that covers the performance of every aspect of their client's operations, such as call volume and call duration. Dialpad's key features are:

- Widget customization:

- The tool has a lot of customization and chart types available, including the only one that offers heat maps;
- Real-time transcriptions: These allow for analytics to analyze client calls and identify areas for improvement;
- IVR analytics

Although Dialpad lacks some features, such as forecasting and scheduling, new updates are made consistently to make the tool as complete as possible. Figure 3.22 shows a dashboard from dialpad.

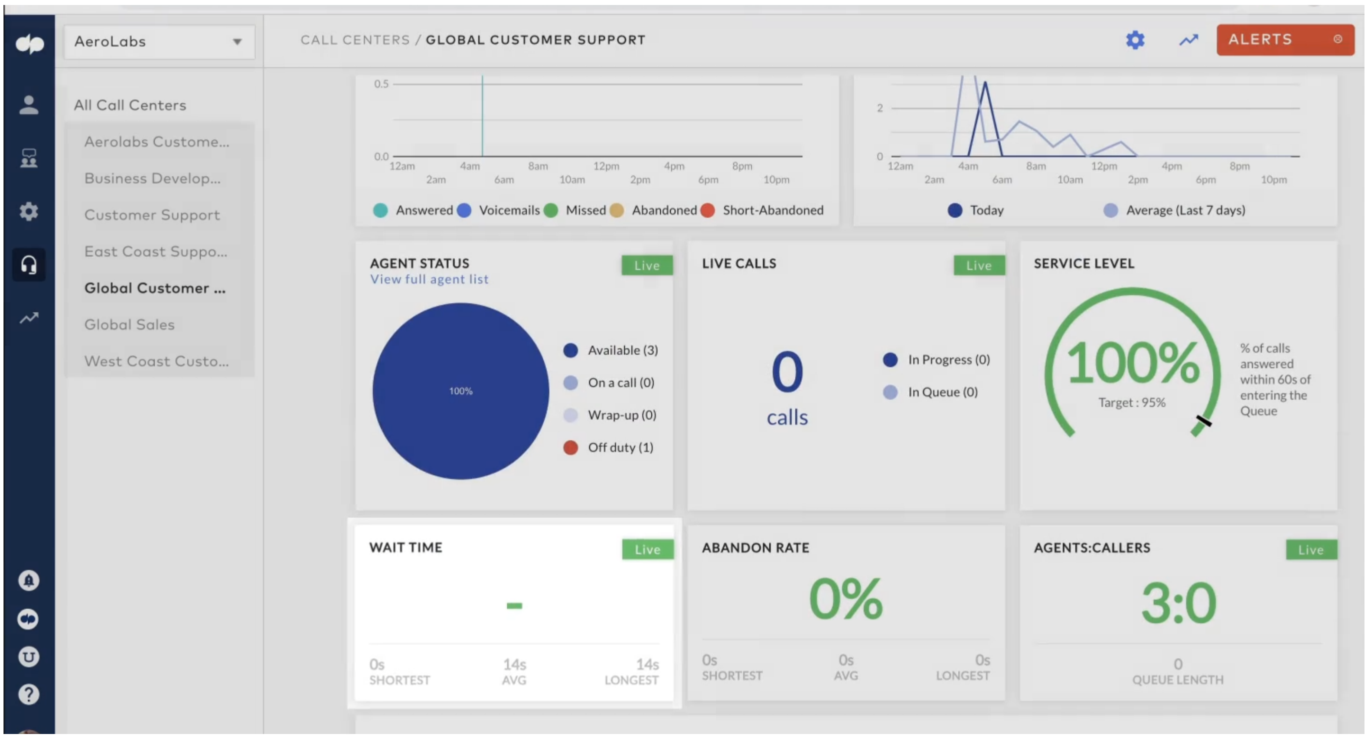


Figure 3.22: Example of a dashboard page in Dialpad. Image taken from [22]

3.11.4 ABC

ABC, as mentioned before, is AlticeLabs' product for UCaaS. Currently, ABC's analytics still have a lot of catching up to do when compared to the major players in the market. Their dashboards are still considered static, with users having no control over which KPIs are displayed and how they are displayed. This project aims to bring ABC's analytics solution closer to industry standards.

Figures 3.23 and 3.24 illustrate the predetermined nature of the charts and the filtering functionality of the ABC system.

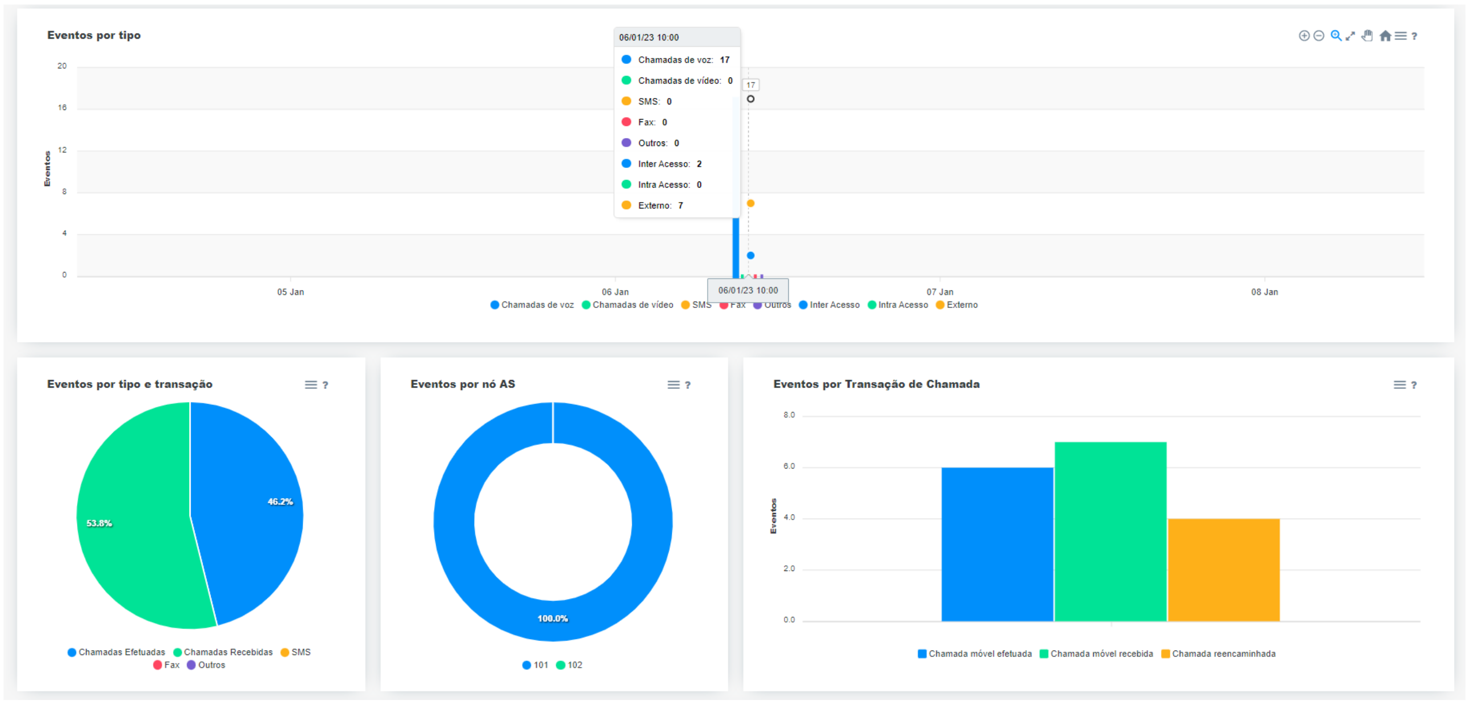
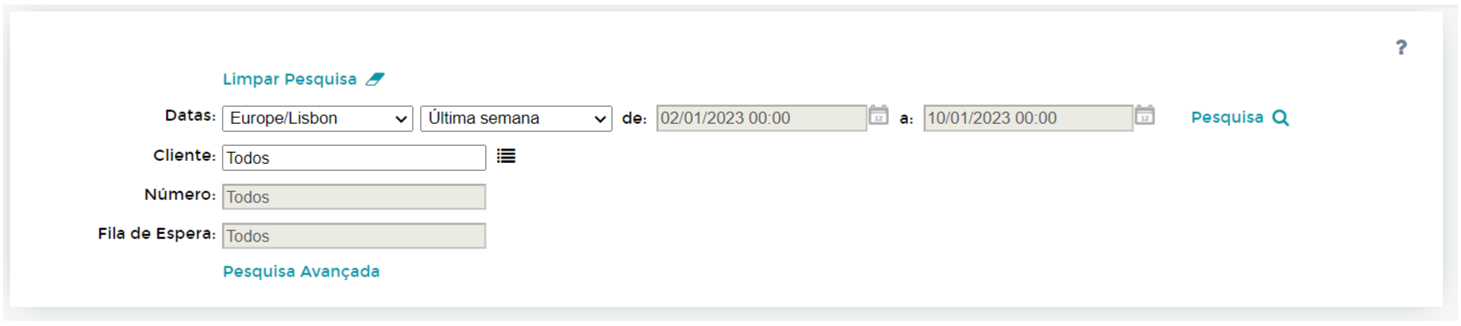


Figure 3.23: Example of some of the charts available in ABC current static dashboard


Limpar Pesquisa

Datas: Europe/Lisbon | Última semana | de: 02/01/2023 00:00 | a: 10/01/2023 00:00 | Pesquisa

Cliente: Todos

Número: Todos

Fila de Espera: Todos

Pesquisa Avançada

Figure 3.24: Filtering on ABC dashboards

3.12 Technologies

This section presents the research on technologies that was conducted to select the ones to use during the development of the project.

3.12.1 Front-End

Framework

JavaScript frameworks offer numerous advantages when developing a large-scale application:

- They are relatively easier to learn and use, with a huge amount of documentation and third-party libraries;
- Single page applications (SPAs) are more efficient in terms of processing, as they require less back and forth between the application and the server. SPAs download everything upfront, only updating the necessary components when needed;
- Better User Interface (UI), as keeping the state of the app in sync with the UI is harder in JavaScript. Every change in the state requires an update in the UI, which can reduce performance with a lot of user interaction. JavaScript frameworks can implement UIs that are guaranteed to be in sync with the internal state of the application;
- JavaScript frameworks have more users, therefore there are better industry standards that promote healthy coding practices.

At AlticeLabs, both Angular.js and React.js are the widely used, so they were the only options considered in this project. The following table compares different parameters between them [51].

Parameters	**Angular.Js**	**React.Js**
Technology Type	Full-fledged MVC framework	JavaScript library (View in MVC)
Concept	Brings JavaScript into HTML Works with real DOM Client-side rendering	Brings HTML into JavaScript Works with the virtual DOM Server-side rendering
Data Binding	Two-way data binding	One-way data binding
Learning Curve	Steep	Moderate
Best Suited For	Highly active and interactive web apps	Larger apps with recurrent variable data
App Structure	Fixes and complicated MVC	Flexible component-based view
Performance	High	High

Table 3.3: React.Js vs Angular.Js

The main parameters taken into account were the *Learning Curve* and the *Best Suited For*. Not only is React.js easier to learn, but the Software Engineering Course also includes a class on it, making the transition to the technology easier and faster. Additionally, React.js excels in areas such as widget updates and component sharing, which will be beneficial for the application.

Chart Library

When selecting the library, it was important to consider various parameters. The first choice was what level of abstraction provided by the library to use. A low abstraction level library like D3.js or a high-level library that abstracts away complexities by providing pre-built components.

While D3.js offers unparalleled customization options and flexibility in creating charts, it comes with a steep learning curve and a huge level of expertise required to work with the library effectively. Starting and maintaining code with D3.js is extremely challenging and time-consuming, making it less accessible for developers who are accustomed to working with high-level libraries.

Considering Alticelabs' preference for high-level libraries, the decision to stick with them aligns with the priority of code maintainability. High-level libraries abstract away much of the underlying complexity, providing developers with pre-built components and intuitive APIs. Additionally, high-level libraries often have comprehensive documentation and a supportive community, facilitating efficient development and reducing maintenance overhead.

The next step was choosing what library to use for the charts. The libraries mostly used in the market were selected and compared to each other. The following parameters were taken into consideration:

Parameters	**Ngx-Charts**	**Ngx-Echarts**	**Angular-Plotly.js**	**Recharts**	**Victory**	**VISX**	**Chart.js**	**High Charts**	**Apex Charts**
Animation	✓	✓			✓	✓	✓	✓	
Responsivity	✓	✓			✓	✓	✓	✓	
Big Community	✓					✓	✓	✓	✓
Client-Side Rendering		✓	✓	✓	✓		✓		✓
Free	✓	✓	✓	✓	✓	✓	✓		✓
Appealing charts design	✓	✓			✓	✓	✓		✓

Table 3.4: Chart Library Features

- Animation and Responsivity $\rightarrow$ These parameters measure if the chart can be interacted with by the user. Zooms, dragging, and other types of interactivity are available if the parameter is checked;
- Large Community $\rightarrow$ A large community means that there are more discussions available and better-defined coding patterns, making it easier to build quality code and fix bugs;
- Client-Side Rendering $\rightarrow$ There are two types of rendering, client-side or server-side. Client-side rendering, although it has worse performance, has a lower load on the server and is preferred by AlticeLabs;

- Free → Some libraries only offer a free trial of their features;
- Appealing charts design → As a dashboard tool, charts need to be visually appealing to the user, as they are the most important part of the dashboard.

Based on the evaluation of the defined parameters, Chart.js emerged as the most suitable choice for developing the dashboard charts. It offers a comprehensive set of features, aligning well with the requirements outlined in the table. One notable advantage of Chart.js is its flexibility in creating custom charts using the tools provided by the library. This allows for the possibility of introducing new chart types in future development, catering to evolving needs and expanding visualization capabilities. Furthermore, Chart.js supports custom plugins, enabling users to contribute and build new features independently without relying solely on library developers. This extensibility fosters a vibrant community around the library, promoting collaboration and the sharing of innovative charting solutions.

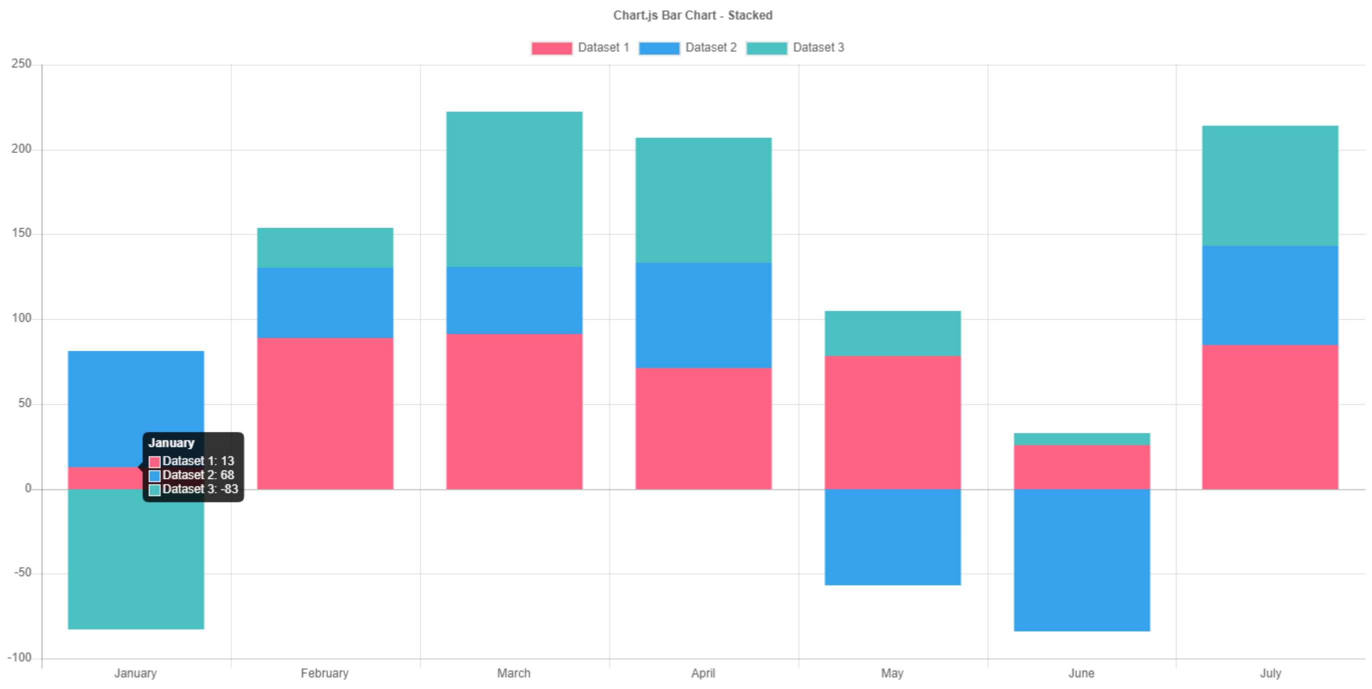


Figure 3.25: Example of Chart.js chart

3.12.2 Back-End

Database

Even though multiple databases are available to use, the one selected was PostgreSQL, as Altice used Postgres based databases, like timescaledb.

REST API

There are several technologies available for the REST API, including Python (Django or Flask), Java (SpringBoot) and Node.js (ExpressJS), which are all used at Altice-Labs and are among the most popular in the world.

Web applications require high concurrency. Node.js, unlike Java, is single-threaded, with a dedicated thread for each request. As the load on the server increases, using multiple threads becomes expensive, as the operating system spends more and more time switching between them. With Node.js, even at full CPU load, the operating system will not break down, and the entire CPU time will be used to service the request. Using a single thread per client will ensure smooth performance, especially in enterprise application development.

ExpressJS, compared to Spring and Django, has a non-blocking, asynchronous I/O model, which allows for better performance while still using a single thread. As ExpressJS is written on JavaScript, it can use the Google v8 engine, which provides better performance without any lag.

Another advantage of using Node.js is that both the front-end and back-end applications will be developed in the same language, reducing the time needed to adapt and the learning curve. Therefore, Node.js will be used for back-end development.

Chapter 4

Architectural Drivers

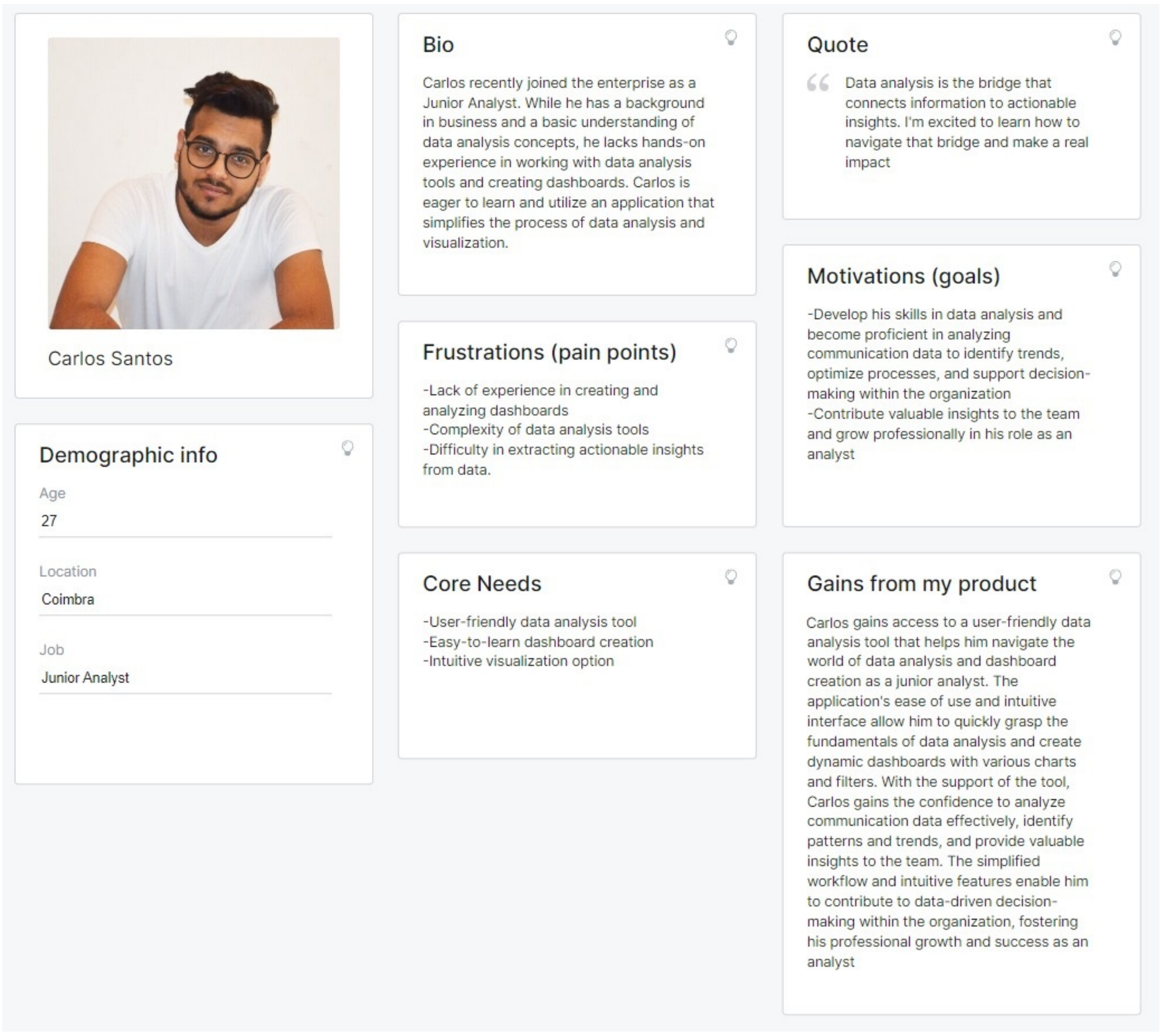
In this section, the architectural drivers that influenced the design and development of our project will be presented. Architectural drivers are factors that shape and guide the overall architecture of a system.

Understanding the architectural drivers of a project is important as they help to define the constraints and requirements that the system must meet. They also help to ensure that the system is aligned with the needs and goals of the organization, and that it is able to adapt to changing requirements and environments.

4.1 Personas

The following subchapter provides an overview of the personas associated with our application. These personas represent different user types and their specific goals and needs. By understanding our target audience, we can design the application to meet their requirements and deliver a tailored user experience.

In figure 4.1 Carlos Santos is represented. Carlos, as a junior analyst, seeks a straightforward dashboard creation process and intuitive visualization options. By simplifying the workflow and offering an intuitive interface it empowers Carlos to navigate the realm of data analysis and visualization with ease. This enables him to contribute valuable insights to his team and supports his professional growth in the field.



Carlos Santos

Bio

Carlos recently joined the enterprise as a Junior Analyst. While he has a background in business and a basic understanding of data analysis concepts, he lacks hands-on experience in working with data analysis tools and creating dashboards. Carlos is eager to learn and utilize an application that simplifies the process of data analysis and visualization.

Quote

“ Data analysis is the bridge that connects information to actionable insights. I'm excited to learn how to navigate that bridge and make a real impact

Demographic info

Age	27
Location	Coimbra
Job	Junior Analyst

Frustrations (pain points)

-Lack of experience in creating and analyzing dashboards
-Complexity of data analysis tools
-Difficulty in extracting actionable insights from data.

Motivations (goals)

-Develop his skills in data analysis and become proficient in analyzing communication data to identify trends, optimize processes, and support decision-making within the organization
-Contribute valuable insights to the team and grow professionally in his role as an analyst

Core Needs

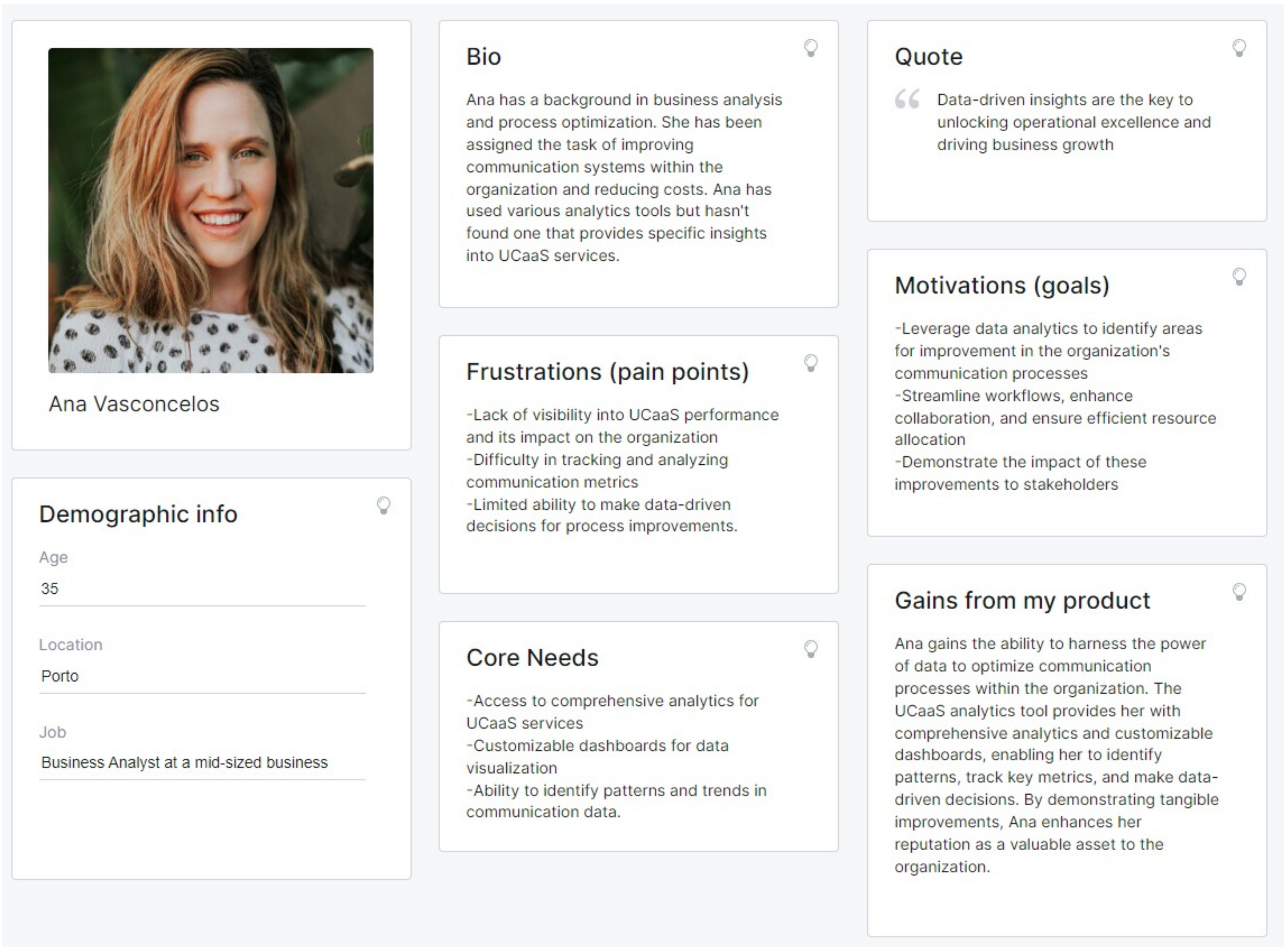
-User-friendly data analysis tool
-Easy-to-learn dashboard creation
-Intuitive visualization option

Gains from my product

Carlos gains access to a user-friendly data analysis tool that helps him navigate the world of data analysis and dashboard creation as a junior analyst. The application's ease of use and intuitive interface allow him to quickly grasp the fundamentals of data analysis and create dynamic dashboards with various charts and filters. With the support of the tool, Carlos gains the confidence to analyze communication data effectively, identify patterns and trends, and provide valuable insights to the team. The simplified workflow and intuitive features enable him to contribute to data-driven decision-making within the organization, fostering his professional growth and success as an analyst

Figure 4.1: Carlos Santos Persona

The persona depicted in Figure 4.2 is Ana Vasconcelos, who seeks comprehensive analytics and customizable dashboards to track communication metrics and drive process improvements. Ana’s frustrations stem from the lack of visibility into UCaaS performance and the difficulties in utilizing data insights to enhance communication systems. By utilizing the capabilities of the platform tool, Ana can streamline workflows, improve collaboration, and effectively demonstrate tangible improvements to stakeholders. This empowers her to drive operational excellence within the organization and make data-driven decisions for process optimization.



Ana Vasconcelos

Bio

Ana has a background in business analysis and process optimization. She has been assigned the task of improving communication systems within the organization and reducing costs. Ana has used various analytics tools but hasn't found one that provides specific insights into UCaaS services.

Quote

"Data-driven insights are the key to unlocking operational excellence and driving business growth

Demographic info

Age	35
Location	Porto
Job	Business Analyst at a mid-sized business

Frustrations (pain points)

-Lack of visibility into UCaaS performance and its impact on the organization
-Difficulty in tracking and analyzing communication metrics
-Limited ability to make data-driven decisions for process improvements.

Motivations (goals)

-Leverage data analytics to identify areas for improvement in the organization's communication processes
-Streamline workflows, enhance collaboration, and ensure efficient resource allocation
-Demonstrate the impact of these improvements to stakeholders

Core Needs

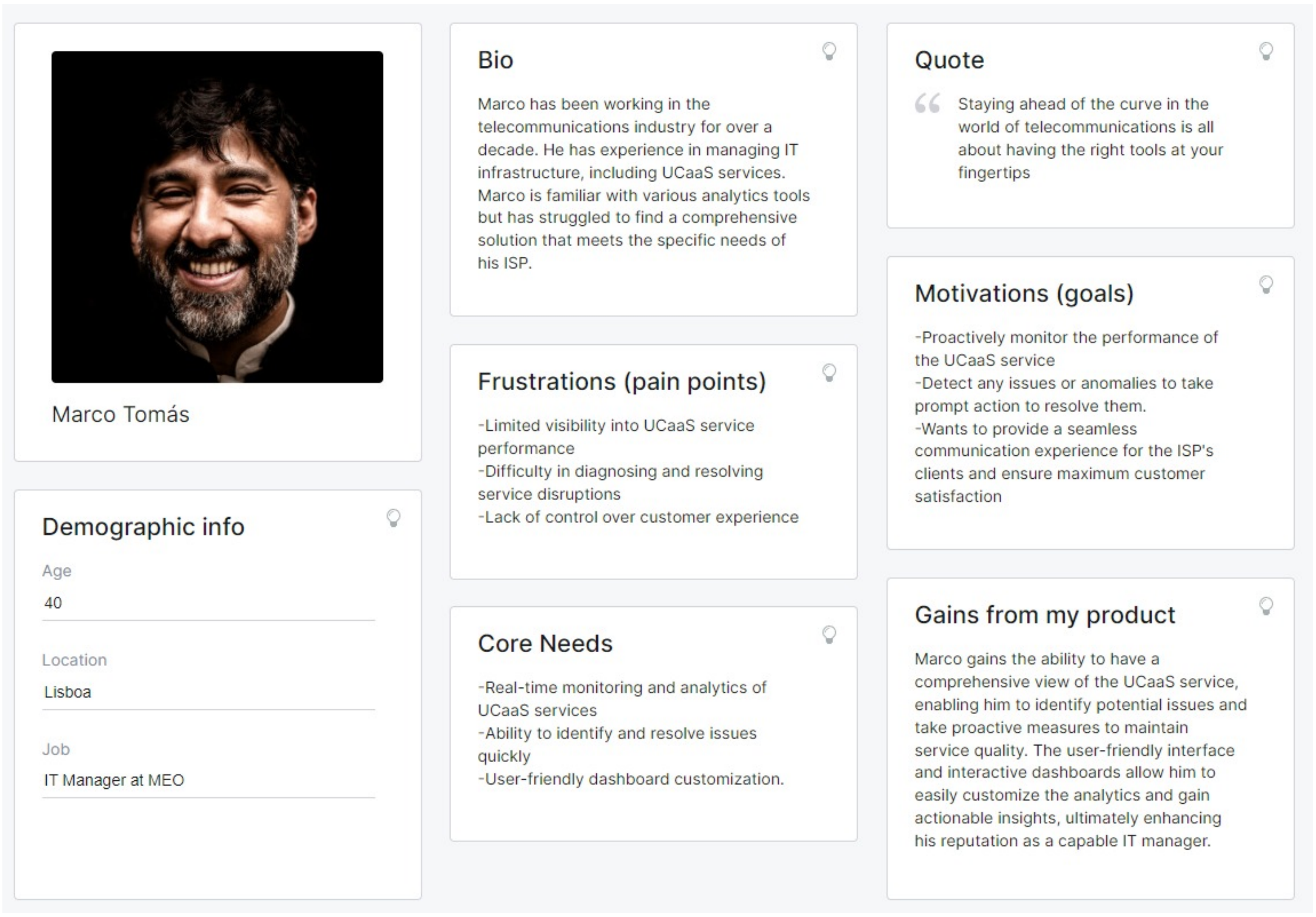
-Access to comprehensive analytics for UCaaS services
-Customizable dashboards for data visualization
-Ability to identify patterns and trends in communication data.

Gains from my product

Ana gains the ability to harness the power of data to optimize communication processes within the organization. The UCaaS analytics tool provides her with comprehensive analytics and customizable dashboards, enabling her to identify patterns, track key metrics, and make data-driven decisions. By demonstrating tangible improvements, Ana enhances her reputation as a valuable asset to the organization.

Figure 4.2: Ana Vasconcelos Persona

The figure represented in Figure 4.3 portrays Marco Tomás, an IT Manager at Meo. The platform offers more than just real-time monitoring and analytics capabilities for Marco. It effectively addresses his needs for proactive issue detection and resolution, as well as the frustrations stemming from limited visibility and control over the UCaaS service. By providing customizable dashboards and widgets, Marco gains the ability to identify potential issues promptly and take necessary action. This ensures that the ISP's customers are consistently satisfied and have an exceptional communication experience.



Marco Tomás

Demographic info

Age
40

Location
Lisboa

Job
IT Manager at MEO

Bio

Marco has been working in the telecommunications industry for over a decade. He has experience in managing IT infrastructure, including UCaaS services. Marco is familiar with various analytics tools but has struggled to find a comprehensive solution that meets the specific needs of his ISP.

Frustrations (pain points)

-Limited visibility into UCaaS service performance
-Difficulty in diagnosing and resolving service disruptions
-Lack of control over customer experience

Core Needs

-Real-time monitoring and analytics of UCaaS services
-Ability to identify and resolve issues quickly
-User-friendly dashboard customization.

Quote

“ Staying ahead of the curve in the world of telecommunications is all about having the right tools at your fingertips

Motivations (goals)

-Proactively monitor the performance of the UCaaS service
-Detect any issues or anomalies to take prompt action to resolve them.
-Wants to provide a seamless communication experience for the ISP's clients and ensure maximum customer satisfaction

Gains from my product

Marco gains the ability to have a comprehensive view of the UCaaS service, enabling him to identify potential issues and take proactive measures to maintain service quality. The user-friendly interface and interactive dashboards allow him to easily customize the analytics and gain actionable insights, ultimately enhancing his reputation as a capable IT manager.

Figure 4.3: Marco Tomás Persona

4.2 Functional Requirements

The section that follows presents a comprehensive examination of the functional requirements for the dashboard platform. These requirements were determined through a collaborative effort, with myself leading the process and working closely with the team to establish the specific functionalities and their corresponding priority levels. Every use-case is clearly outlined and cross-referenced in Appendix A, providing a clear and organized framework for the analysis of the dashboard platform's functionalities. In order to write the use cases, a combination of the shortened *one-column style* and the *two-column style* for the main flow was utilized, following Cockburn's guidelines [20].

4.2.1 Use Case Diagram

This section presents the use case diagram for the dynamic dashboard application. Figure 4.4 showcases the connections between the use cases. It provides a visual representation of how different functionalities and features interact within the application, offering a comprehensive overview of the system's behavior and functionality.

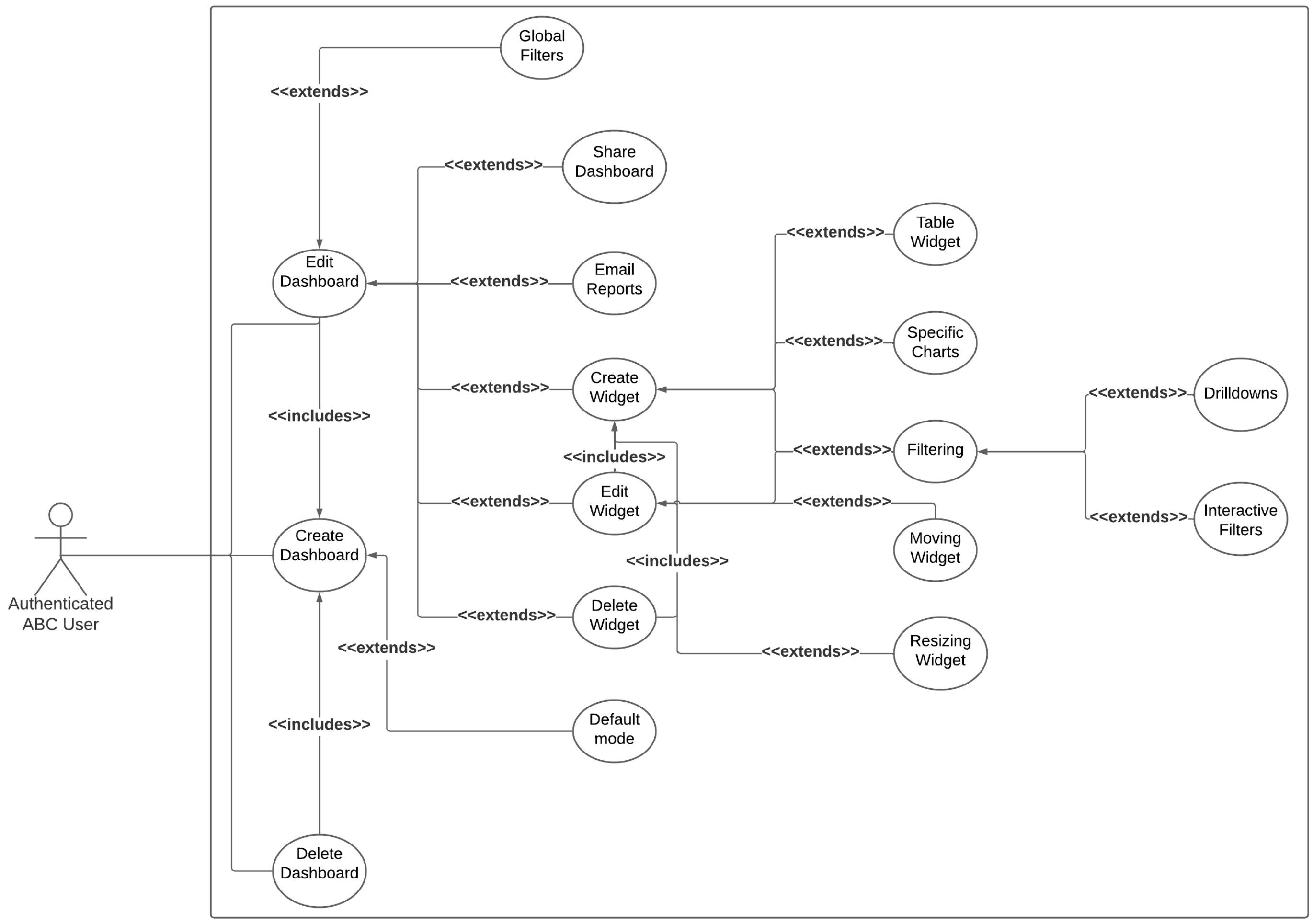


Figure 4.4: Use Case Diagram

Dashboard

Users have the ability to create customized dashboards, allowing them to tailor the dashboard to their specific needs. This caters to different requirements such as real-time monitoring for Marco, analytics for Ana, or practice and learning for Carlos. Additionally, users can modify and customize existing dashboards, empowering them to refine their dashboards to suit their preferences. This is particularly beneficial for Marco and Ana, who can adapt the dashboards to track performance and make informed decisions, and for Carlos, who can experiment and learn by modifying existing dashboards. Furthermore, users have the option to remove unwanted or outdated dashboards, providing flexibility in managing their dashboard inventory and keeping it organized. Although not explicitly mentioned as a need for any persona, the ability to delete dashboards benefits all users by enabling efficient management of their dashboard collection.

Users can apply global filters that impact all widgets on a dashboard, providing a holistic view of the data. This feature allows Marco to analyze the impact of each filter on service performance, Ana to evaluate overall communication metrics across the organization, and Carlos to explore the effects of different filters on multiple widgets for comprehensive analysis.

Furthermore, users can share their dashboards with others, which is particularly relevant for Marco and Ana. Marco can share dashboards with clients to showcase service performance, while Ana can collaborate with stakeholders and demonstrate the impact of communication improvements. This sharing capability facilitates effective communication and collaboration within and outside the organization.

The option to set a default mode during dashboard creation establishes a preferred starting point for analysis or presentation. This benefits all personas, especially Marco and Ana, who can focus on specific metrics or views, and Carlos, who can use it as a guide to create new widgets and dashboards. Generating and sending automated email reports from dashboards allows users to schedule and share regular reports with stakeholders. This is advantageous for all personas, particularly Marco and Ana, who may need to share performance metrics and analysis with clients or stakeholders, enhancing communication and providing updated insights.

These additional features (default mode and email reports) provide users with enhanced capabilities and convenience, contributing to a more efficient and effective user experience.

Widget

When creating a widget, users have the flexibility to select desired metrics by dragging and dropping them onto the appropriate axis based on the chosen chart type. This feature caters to the different needs of each persona, as different chart types have specific requirements. Additionally, users can choose from a range of pre-defined colors for their widgets, simplifying the process and enhancing usability.

After widget creation, users can modify and customize them on their dashboards. This functionality enables users to fine-tune the widgets, displaying relevant information and metrics based on their specific needs. Marco can customize widgets to monitor services, Ana can analyze communication metrics, and Carlos can experiment with different visualization options. Users also have the ability to delete widgets, providing them with flexibility in managing their dashboard content.

Applying filters to analyze specific subsets of data is essential for all personas.

Marco can troubleshoot service disruptions, Ana can identify areas for improvement in communication processes, and Carlos can practice analyzing data subsets to extract valuable insights. Users can apply filters by dragging metrics onto the filter dropzone or interactively by clicking on values within widgets. This use case benefits Carlos in particular, as it supports his hands-on learning experience by allowing him to dynamically adjust filters and observe the resulting data insights.

During widget editing, users can move and resize widgets, optimizing the visual layout of their dashboards and improving their overall experience with the application. This feature allows users to arrange widgets in a manner that suits their preferences, ensuring a more efficient and intuitive dashboard design.

Users can drill down into detailed information from summary views, enabling more in-depth analysis and identification of specific insights. This use case is valuable for all personas, as it facilitates thorough investigation and understanding of communication data, empowering users to make more informed decisions.

The ability to add table-based widgets to dashboards provides a structured and organized representation of data. Users can compare values, track trends, and perform detailed analysis using tables. This use case benefits all personas, allowing them to examine specific data points in a tabular format, enhancing their data exploration capabilities.

4.2.2 Use Case Priority

Through collaborative discussions with the team, all of the use cases were carefully prioritized using the Must have, Should have, Could have and Won't have (MoSCoW) method. During the prioritization process, the "Must Have" functionalities were identified as the highest priority, ensuring that they are essential for the successful implementation and usability of the dashboard platform in the first deployment of the application.

Must Have:

The must-have use cases are essential and directly address the core needs and frustrations of all three personas. These use cases provide the foundation for customizing and analyzing data, enabling all personas to monitor, analyze, and troubleshoot the ABC services effectively. These are:

- FR01: Create Dashboard
- FR02: Edit Dashboard
- FR03: Delete Dashboard

- FR04: Create Widget
- FR05: Edit Widget
- FR06: Delete Widget
- FR07: Filtering
- FR08: Global Filters

Should Have:

The should-have use cases enhance the personas' experience and provide additional value without compromising the core functionality. These use cases contribute to collaboration, hands-on learning, and improved visual presentation of data, benefiting the personas in their respective roles. They are:

- FR09: Share Dashboard
- FR13: Interactive filtering
- FR15: Moving Widget
- FR16: Resizing Widget

Could Have:

The could-have use cases offer additional flexibility and customization options to further enhance the user experience. These use cases allow for more specialized data visualization, personalized starting points, and deeper data exploration, providing added value for users who require advanced analysis and presentation capabilities. They are:

- FR10: Specific charts for widgets
- FR11: Default mode
- FR12: Table Widget
- FR14: Drilldowns
- FR21: Email Reports

Won't Have:

The won't-have use cases are deemed less critical and have been excluded from the prioritization. While these features may offer convenience for data transfer or backup purposes, they are not essential for the personas' immediate needs and can be considered as potential future enhancements. They are:

- FR17: Importing Widget

- FR18: Exporting Widget
- FR19: Importing Dashboard
- FR20: Exporting Dashboard

4.3 Non-Functional Requirements

In this section, we will discuss the non-functional requirements of this project. These requirements are critical to the success of the project as they ensure the quality and reliability of the system. Non-functional requirements include aspects such as performance, security, usability, and maintainability, among others. For each quality attribute a scenario was created with the following characteristics, following the Quality Attribute Scenario [48]:

- Stimulus: Condition to be considered when it arrives at the system;
- Source: Entity generating the stimulus;
- Environment: Circumstances under which the stimulus occurs e.g. normal operation, stress conditions, fault etc;
- Artifact: Part of the system that is stimulated;
- Response: System activity undertaken as a result of the stimulus arriving;
- Measure: How well the response should satisfy the requirement.

In order to prioritize essential qualities for the application, two key attributes were identified as priority: usability and maintainability. Given that the dashboard application caters to a diverse user base with varying levels of experience, usability takes precedence to ensure an intuitive and user-friendly interface. Additionally, with the objective of facilitating future development beyond the internship period, emphasis is placed on maintainability to enable smooth code comprehension and adoption by new developers. These priorities ensure an accessible and sustainable application throughout its life cycle.

Usability

Three scenarios were defined for usability:

Source of Stimulus	User
Stimulus	User needs to perform a specific task within the application efficiently.
Environment	Normal operation
Artifact	Dashboard Tool
Response	Intuitive and streamlined user interface
Response Measure	The user completes a list of tasks on the application in less than 2 minutes each, with at least a 85% accuracy rate

Table 4.1: Scenario 1 of Usability

Source of Stimulus	User
Stimulus	Novice user wants to learn the basic features and functionalities of the application and navigate through the interface
Environment	Normal operation
Artifact	Dashboard Tool
Response	Provide a guided tour or tutorial that introduces the key features, navigation elements, and basic operations of the application
Response Measure	The user successfully completes the guided tour with an average completion time of less than 5 minutes and is able to reproduce it with at least 95% success rate

Table 4.2: Scenario 2 of Usability

Source of Stimulus	User
Stimulus	User wants to personalize the application based on their preferences and workflow
Environment	Normal operation
Artifact	Dashboard Tool
Response	Enable users to customize the interface layout
Response Measure	Customization of UI and widgets takes less than 3-clicks

Table 4.3: Scenario 3 of Usability

Table 4.1, 4.2 and 4.3 define the usability scenarios. During normal operation the user must be able to learn and use the application efficiently and customize it to his needs. In order to measure if these goals were obtained a System Usability Scale (SUS) was done. The results and more information about SUS is found in chapter 6 .

Maintainability

For maintainability 3 scenarios were defined:

Source of Stimulus	Developer
Stimulus	Developers want to create modular and reusable code components for efficient maintenance
Environment	Normal development environment
Artifact	Codebase of the application
Response	Design and implement code with a modular architecture, separating concerns and creating reusable components or libraries.
Response Measure	The fontend codebase should have a modular structure, with at least 80% of code components identified as reusable

Table 4.4: Scenario 1 of Maintainability

Source of Stimulus	Developer
Stimulus	Developers want to write tests easily and effectively for code maintenance
Environment	Normal development environment
Artifact	Codebase of the application
Response	Design code with clear separation of concerns and proper decoupling to facilitate unit testing
Response Measure	The API codebase should have a line coverage of at least 80%

Table 4.5: Scenario 2 of Maintainability

Source of Stimulus	Developer
Stimulus	Developers want to have comprehensive and up-to-date documentation for code maintenance
Environment	Normal development environment
Artifact	Codebase of the application
Response	Document code with clear comments, API documentation, and high-level architectural documentation
Response Measure	Every component prop and endpoint response and request body needs to be documented

Table 4.6: Scenario 3 of Maintainability

The three scenarios above 4.4, 4.5 and 4.6 scenarios emphasize three key attributes of maintainability in software development: modularity and reusability, testability, and documentation. These attributes enable efficient maintenance by facilitating code updates, reducing bugs through testing, and enhancing code understanding.

4.4 Restrictions

The following section provides an overview of the restrictions that have been imposed on the project. Project restrictions refer to any limitations, constraints or factors that have been placed on the project and its objectives, goals, and deliverables.

Business restrictions

Business restrictions may not directly affect the architecture of the project, however, they can still have an impact on it. One example of this is a reduction in the time allocated for architectural definition. The business restrictions identified for the project and internship include:

- **BR1 - Development time**
  – Rational: The development time of the system should not exceed the duration of the internship;
  – Flexibility points: Extensions to the duration of the internship;
  – Viable alternatives: None.

Technical restrictions

Technical restrictions can have a greater impact on the architecture of the project than business restrictions as they can limit the choices for technologies, protocols, programming languages, etc. The technical restrictions identified for the project include:

- **TR1 - ABC Authentication**
  – Rational: The authentication should be done via ABC platform;
  – Flexibility points: None;
  – Viable alternatives: Use 3 hardcoded logins as a temporary solution due to the unavailability of necessary endpoints for ABC platform authentication.

- **TR2 - The system must contain a RESTful API**
  – Rational: The front end application needs a REST API in order to communicate with the database;
  – Flexibility points: None;
  – Viable alternatives: None.

- **BR2 - Internationalization**
  – Rational: The application should be easily translated to various languages via a JSON file;
  – Flexibility points: None;
  – Viable alternatives: None.

4.5 C4 Model

In this section, we will introduce the C4 model for software architecture. The C4 model is a visual notation for representing the structural and behavioral aspects of software systems in a clear and concise manner.

The C4 model consists of four levels of abstraction, each representing a different aspect of the system: Context, Container, Component, and Code [13].
The Code Diagram was not done as such a low level of detail is not needed for this project.

In the following sections, we will explore each of the levels of the C4 model in more detail, and how they can be used to understand and communicate the architecture of a software system. The full architecture of the system is displayed, even if some features will end up not being developed in the context of this thesis, for example, the mobile portability.

4.5.1 Context Diagram

In this section the components of the Context diagram are presented and explained.

A Context diagram is typically the highest level of abstraction in a software architecture model, and it provides a broad overview of the system and its place within the larger context. It is used to communicate the overall goals and objectives of the system, and to identify the key actors and external dependencies that shape its behavior and functionality.

The figure below, figure 4.5, illustrates the intended use of the dashboard application by its users and the supporting role of a backend application in enabling the desired features.

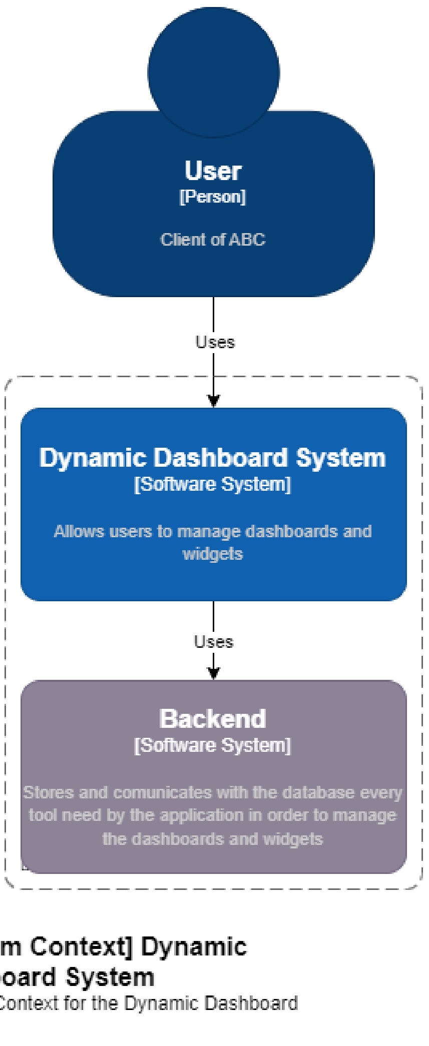

User
[Person]
Client of ABC
Uses
Dynamic Dashboard System
[Software System]
Allows users to manage dashboards and widgets
Uses
Backend
[Software System]
Stores and comunicates with the database every tool need by the application in order to manage the dashboards and widgets
[System Context] Dynamic Dashboard System
System Context for the Dynamic Dashboard System

Figure 4.5: Context Diagram of the C4 Model

4.5.2 Container Diagram

In this section the components of the Container Diagram are presented and explained.

A Container diagram is a level of abstraction below the Context diagram, and it provides a more detailed view of the system's deployment environment. It is used to communicate the physical and logical components that make up the system's runtime environment, and to show the relationships and dependencies between these components.

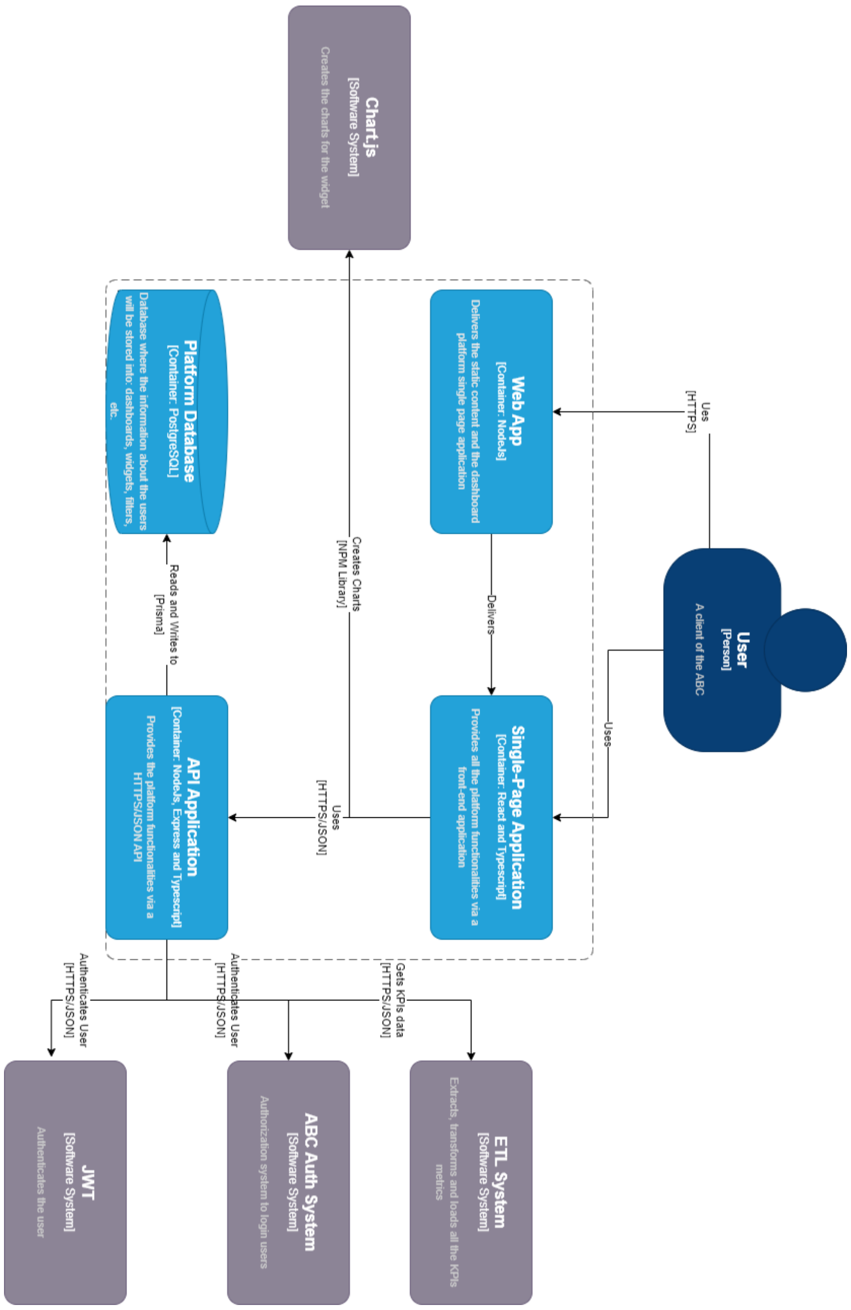


Chart.js
[Software System]
Creates the charts for the widget
User
[Person]
A client of the ABC
Ues
[HTTPS]
Uses
Web App
[Container: Node.Js]
Delivers the static content and the dashboard platform single page application
Delivers
Single-Page Application
[Container: React and Typescript]
Provides all the platform functionalities via a front-end application
Creates Charts
[NPM Library]
Uses
[HTTPS/JSON]
Platform Database
[Container: PostgreSQL]
Database where the information about the users will be stored into: dashboards, widgets, filters etc.
Reads and Writes to
[Prisma]
API Application
[Container: Node.Js, Express and Typescript]
Provides the platform functionalities via a HTTPS/JSON API
Gets KPIs data
[HTTPS/JSON]
Authenticates User
[HTTPS/JSON]
Authenticates User
[HTTPS/JSON]
ETL System
[Software System]
Extracts, transforms and loads all the KPIs metrics
ABC Auth System
[Software System]
Authorization system to login users
JWT
[Software System]
Authenticates the user

Figure 4.6: Container Diagram of the C4 Model

Figure 4.6 illustrates the composition of the dashboard system, which consists of four containers: a server-side web application, a client-side single-page application (SPA), a server-side API application, and a database. The web application serves static content for the SPA, which is built using React.JS and utilizes Chart.js as an external container to display charts. The SPA uses a JSON/HTTP API developed in Node.js to communicate with the ABC authentication system for login and the ETL system to obtain KPIs data. All data from the API is stored in the database.

4.5.3 Component Diagram

In this section the component diagram is presented and explained.

The Component diagram is a visual representation of the high-level components that make up a system and the relationships between them. These components can include modules, libraries, frameworks, and other logical units of functionality that are used to implement the system's functionality.

A Component diagram is a level of abstraction below the Container diagram, and it provides a more detailed view of the system's functional components. It is used to communicate the overall structure and organization of the system, and to show how the components are related and interact with one another.

Figure 4.7 illustrates the components comprising the REST API. The Authentication Controller plays a crucial role in user authentication, ensuring the security of the API by preventing unauthorized access. It leverages the ABC authentication system for user login and utilizes JSON Web Tokens (JWT) for query authentication.

The Dashboard Controller serves as a central hub for managing dashboards, utilizing the Statistic Controller to retrieve a comprehensive list of available KPIs. Additionally, it interacts with the Widget Controller to handle widget management within specific dashboards. The Widget Controller, in turn, relies on the Color Controller, which facilitates the listing of available color ranges for use in widgets. Furthermore, the Analytics Controller plays a crucial role by loading and transforming data into a format compatible with the Chart.JS library, enabling the smooth visualization of data in charts.

This component-based architecture ensures the effective and secure operation of the REST API, allowing for seamless authentication, dashboard management, widget customization, and data visualization functionalities.

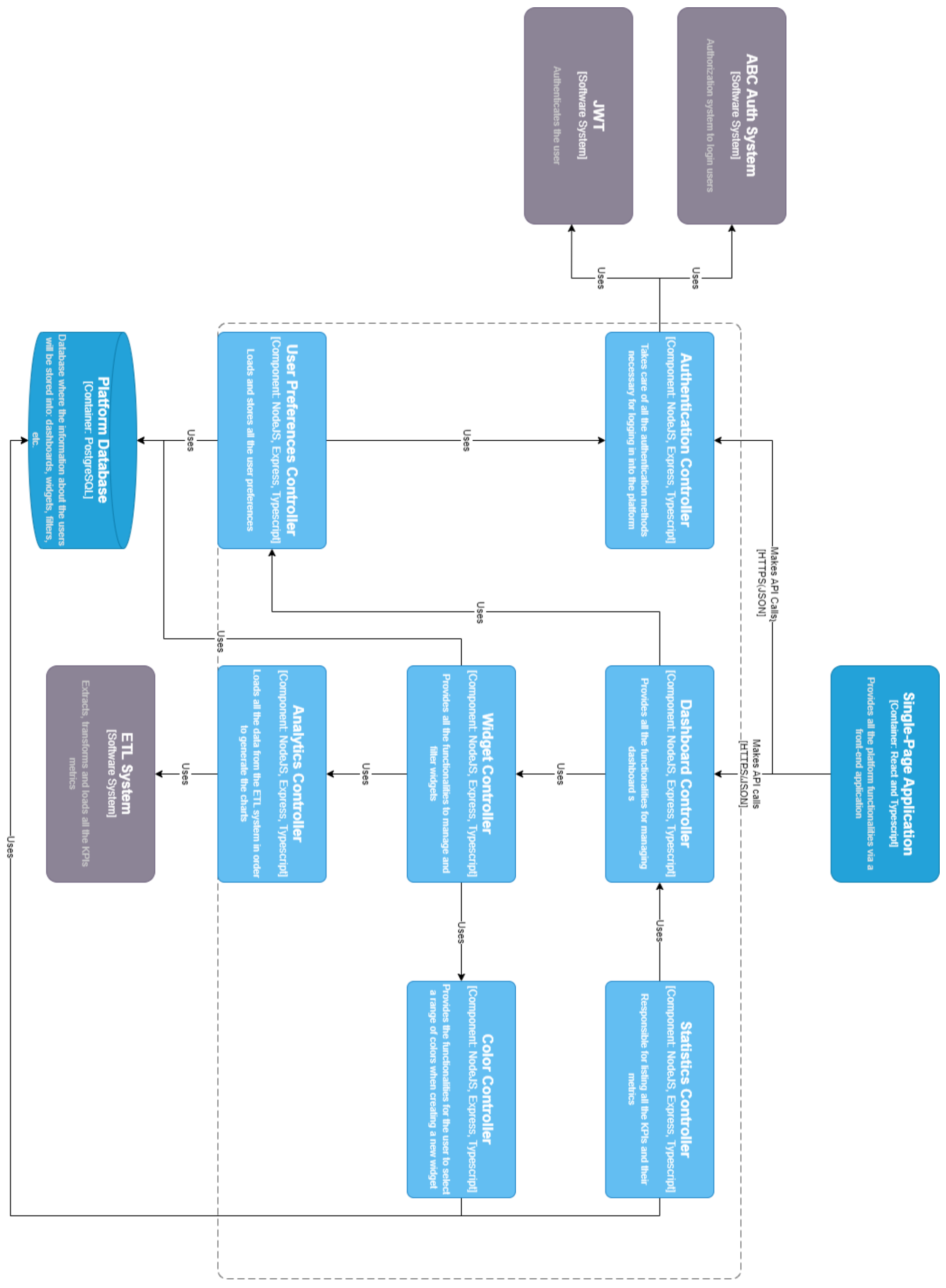

ABC Auth System
[Software System]
Authentication system to login users
JWT
[Software System]
Authenticates the user
Uses
Uses
Authentication Controller
[Component: NodeJS, Express, Typescript]
Takes care of all the authentication methods necessary for logging in into the platform
User Preferences Controller
[Component: NodeJS, Express, Typescript]
Loads and stores all the user preferences
Platform Database
[Container: PostgreSQL]
Database where the information about the users will be stored into: dashboards, widgets, filters, etc.
Makes API Calls
[HTTPS/JSON]
Single-Page Application
[Container: React and Typescript]
Provides all the platform functionalities via a front-end application
Dashboard Controller
[Component: NodeJS, Express, Typescript]
Provides all the functionalities for managing dashboard s
Widget Controller
[Component: NodeJS, Express, Typescript]
Provides all the functionalities to manage and filter widgets
Analytics Controller
[Component: NodeJS, Express, Typescript]
Loads all the data from the ETL system in order to generate the charts
ETL System
[Software System]
Extracts, transforms and loads all the KPIs metrics
Statistics Controller
[Component: NodeJS, Express, Typescript]
Responsible for listing all the KPIs and their metrics
Color Controller
[Component: NodeJS, Express, Typescript]
Provides the functionalities for the user to select a range of colors when creating a new widget
Uses

Figure 4.7: Component Diagram of the C4 Model

4.6 Mockups

The following section presents an overview of the main prototypes developed for the platform, offering visual representations of the platform's design and functionality. These prototypes serve as tangible examples of the user interface, user experience, and key features outlined in previous sections. The showcased prototypes highlight the essential elements and functionalities of the platform, providing readers with a comprehensive understanding of its design and usability. For additional mockups, please refer to Appendix B. The mockups were created using Figma, adhering to the Altice brand design guidelines.

4.6.1 Altice Guideline

The following subsection provides a detailed exploration of the Altice brand guidelines that were followed during the design and development of the platform. These brand guidelines serve as a framework to ensure consistency in visual identity and user experience across all Altice products and services. By adhering to these guidelines, the platform aligns seamlessly with the Altice brand and creates a cohesive and recognizable user experience.

The brand guidelines encompass various aspects, starting with the guidelines for the logo. The guidelines specify the options for using a horizontal or vertical logo, ensuring a designated space around the logo, and defining the minimum and maximum sizes. Additionally, the guidelines stipulate that the logo should be used on white or black backgrounds, which has been implemented accordingly, particularly in the platform's sidebar.

Typography is another critical aspect covered in the brand guidelines. The guidelines mandate the use of the "Altice" font family and specify "Montserrat" as the designated font for web applications. Thus, the chosen font for the platform is the Montserrat family. Furthermore, in adherence to the guidelines, the font color used throughout the platform is exclusively black or white.

The brand guidelines also provide a defined color palette for the application. The primary and secondary palettes are depicted in Figure 4.8 and Figure 4.9, respectively.

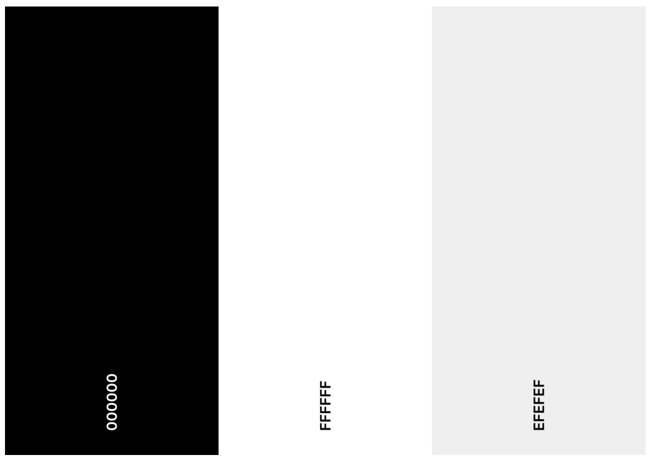


Figure 4.8: Primary palette

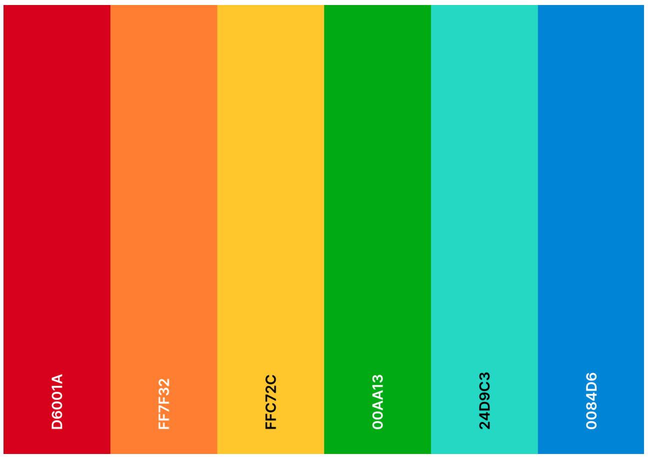


Figure 4.9: Secondary palette

4.6.2 User Experience

As highlighted in Section 3.4, there are various approaches to designing a dashboard. After discussing with the team, it was determined that the application would incorporate an open layout feature, enabling users to freely select the size and position of their widgets, as this design decision aligns with Fitts's Law. By allowing users to customize the size and position of widgets, the platform enhances the overall user experience and enables personalized interface configurations.

The introduction of a multipage system in the dashboard offers several advantages. Firstly, it improves navigation within the widgets by providing users with the ability to divide and organize their content across multiple pages. This enhances the overall usability and organization of the dashboard, particularly for users with a large number of widgets. Additionally, the multipage system contributes to the performance optimization of the dashboard, as it avoids loading all widgets simultaneously. This ensures a smoother user experience, with faster loading times and improved responsiveness. Furthermore, the multipage system benefits the readers of the dashboard, as they can focus on specific pages and analyze the content without being overwhelmed by a large number of widgets simultaneously.

Although the MVP currently consists of a single page, the underlying architecture and design of the application are capable of accommodating multiple pages. This scalability allows for future expansion and the addition of more pages as the platform evolves, providing flexibility and adaptability to meet evolving user needs and requirements.

4.6.3 Mockup Design and Explanation

In this subsection a detailed explanation of the mockups will be provided, accompanied by an explanation of the design choices.

Main Page

The main page of the application is depicted in Figure 4.10. It features a well-designed layout comprising two primary sections. The first section is a collapsible sidebar that expands upon hovering and contracts when not in use. The second section is the main content area, which occupies the majority of the screen. This layout optimizes the space available for content while maintaining an aesthetically pleasing interface.

One notable aspect of this design is the flexibility it offers to users. They have the option to choose the placement of the sidebar based on their preferences. Figure B.2 illustrates the mockup with the navbar positioned on the left side. This concept was inspired by a Dribbble dashboard mockup created by Eugenie Rumiantseva [55]. The overall goal of this layout is to provide a visually appealing and customizable user experience.

On this page, users have the ability to create a new dashboard by hovering over the "+" button and selecting the "new" option. Their created and shared dashboards will be presented in a grid format, organized based on the order of creation. Users can then open, edit, and delete their created dashboards as needed.

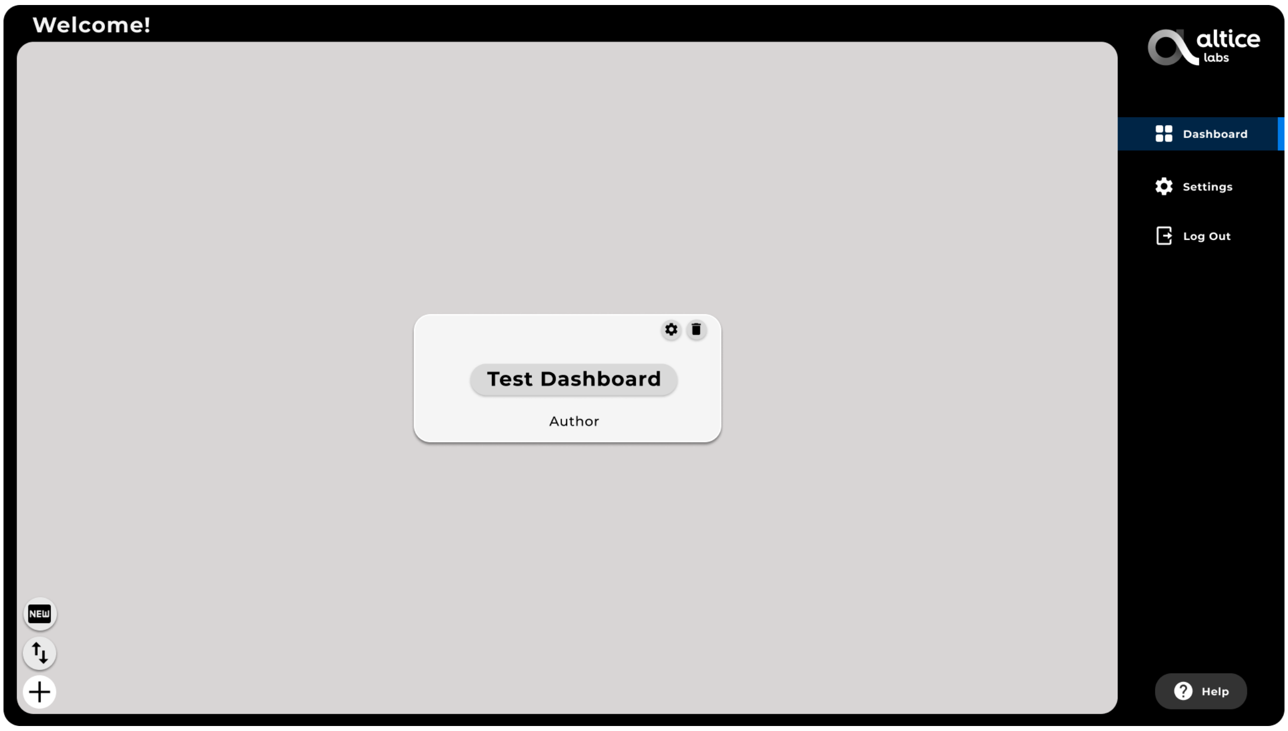


Figure 4.10: Mockup of the Main Page

Creating a Widget

The provided mockups (Figures 4.11, 4.12, and 4.13) illustrate the process of creating a new widget within the application. These images showcase the dashboard area, which consists of a background where the widgets will be displayed, as well as three tabs. The tabs offer users the ability to create, edit, and filter their widgets. It's worth noting that the position of these tabs can be customized through the settings, as shown in Figure B.3, allowing users to personalize their experience.

The inclusion of these tabs aligns with the approach taken by various other dashboard BI tools, such as PowerBI, to provide users with a familiar and consistent user experience. Each tab serves a distinct purpose: one for listing Key Performance Indicators (KPIs) and their associated metrics, another for creating and editing widgets. In the widget creation tab, users can select the desired chart type and utilize a drag-and-drop functionality to add their desired metrics to the corresponding axes.

Figure 4.13 showcases the configuration of a regular axis widget, such as a line chart or bar chart. On the other hand, Figure B.4 demonstrates the creation of a horizontal bar chart with an inverted axis display. Additionally, Figure B.5 exhibits the creation of a line-column axis widget, where separate axes are utilized for the bar and line elements. Lastly, Figure B.6 presents the creation of a pie chart, allowing users to select the desired value to display and specify how the pie should be divided. Furthermore, users have the option to choose a color range for their widgets.

In the third tab, users can leverage drag-and-drop functionality to add both global filters and widget-specific filters (applicable only when creating or selecting a widget). By dragging the desired metric from the KPI section, users can define the filtering criteria based on specific values. The global filter section provides additional filtering options, such as start and end date filters, along with a recurrence filter that determines the data's periodicity.

In Figure 4.14, the displayed widget on the dashboard demonstrates various user actions that can be performed on it. The user has the flexibility to resize the widget according to their preference and relocate it to their desired position on the dashboard. Additionally, the widget offers options to apply filters, edit its contents, and delete it altogether. These functionalities empower users to customize their dashboard layout and interact with the widgets in a dynamic and personalized manner.

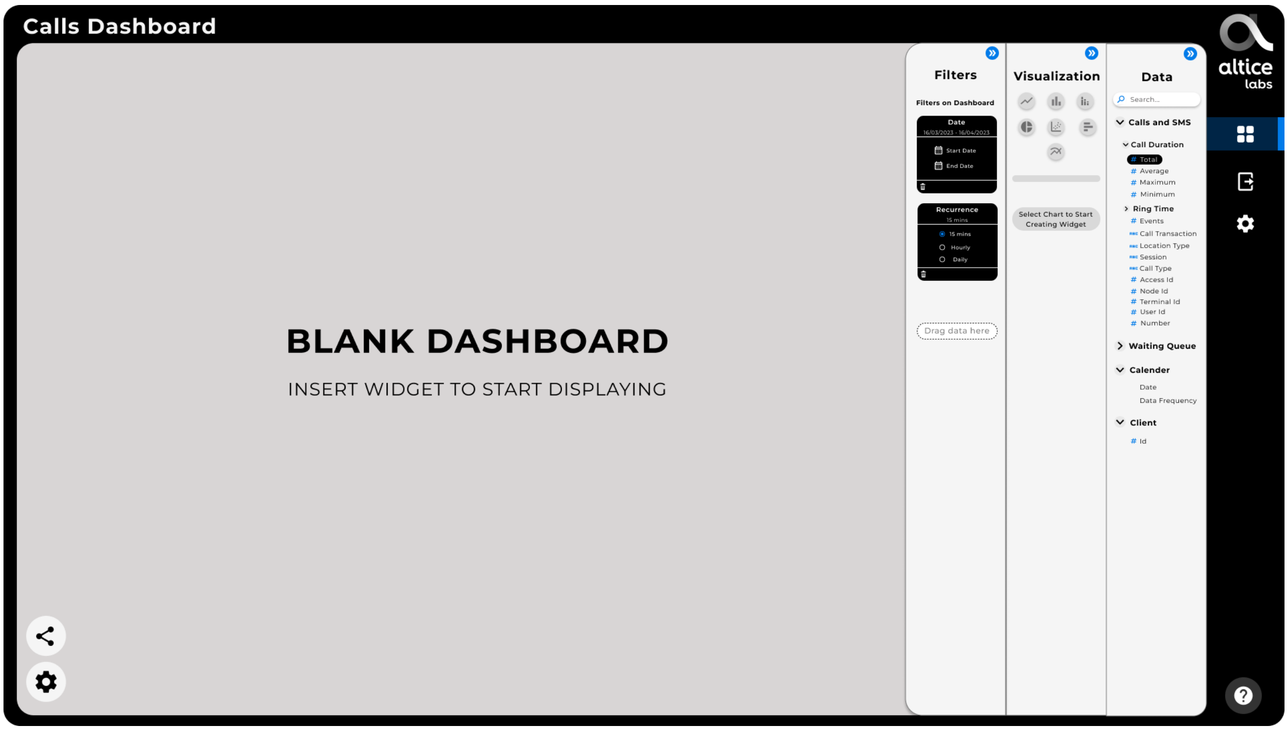


Figure 4.11: Mockup of Creating a new Widget

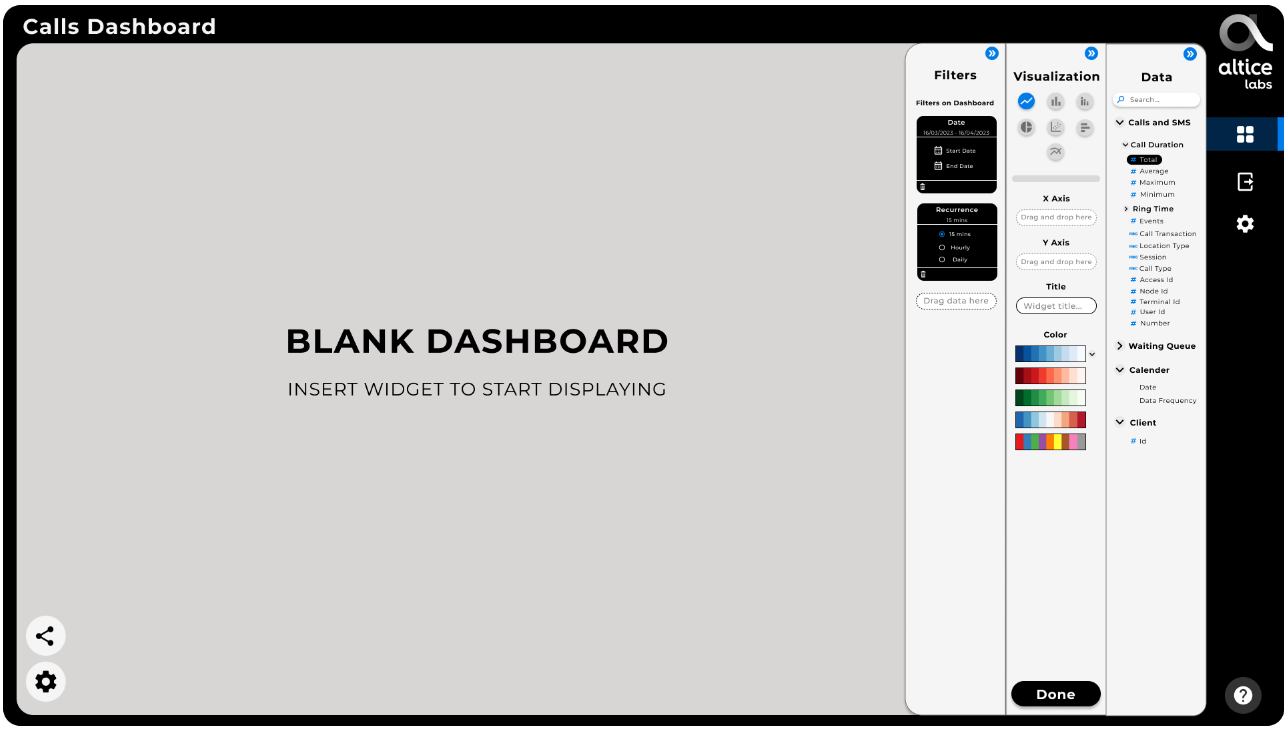


Figure 4.12: Mockup of Creating a new Widget part 2

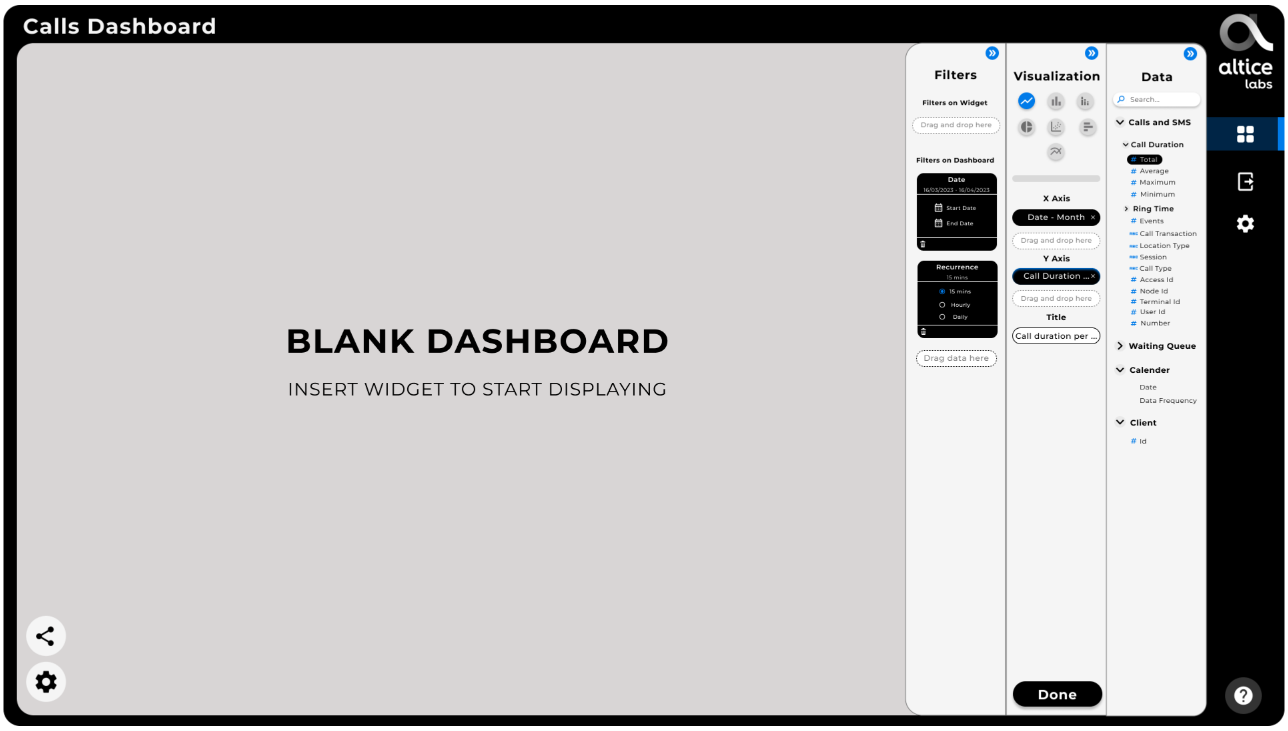


Figure 4.13: Mockup of Creating a new Widget part 3

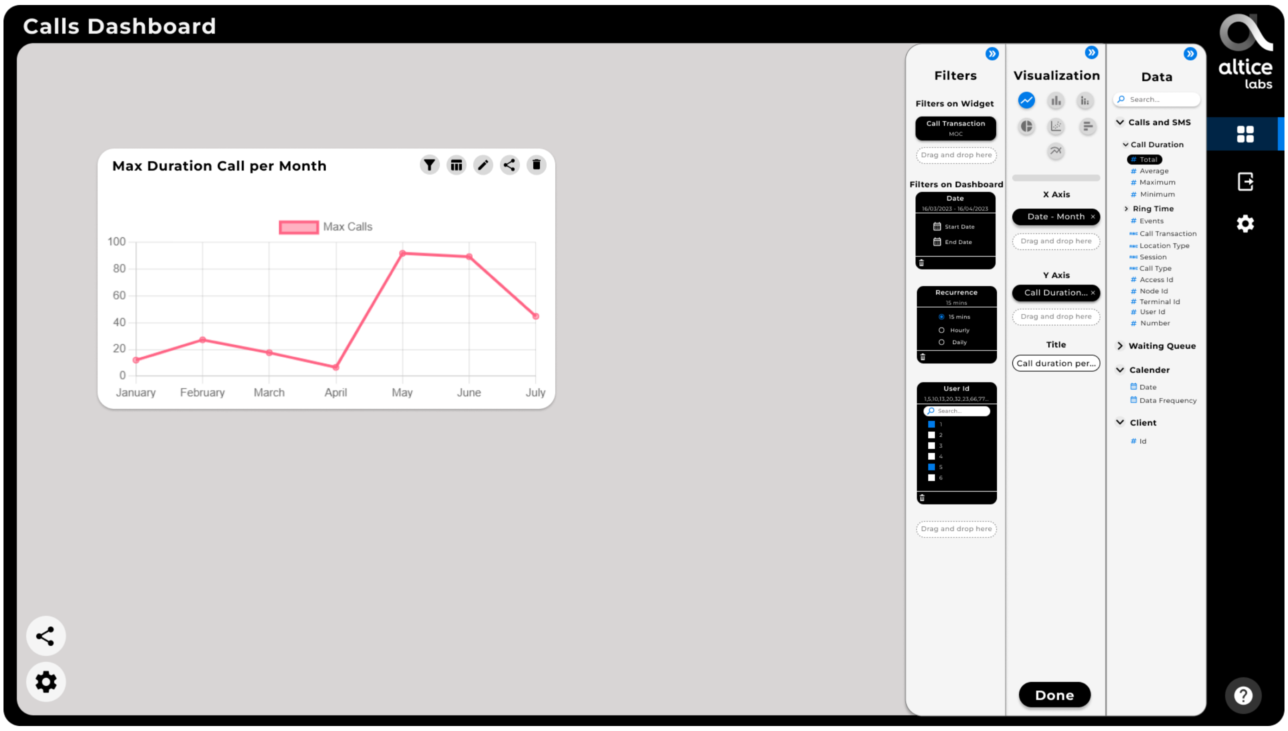
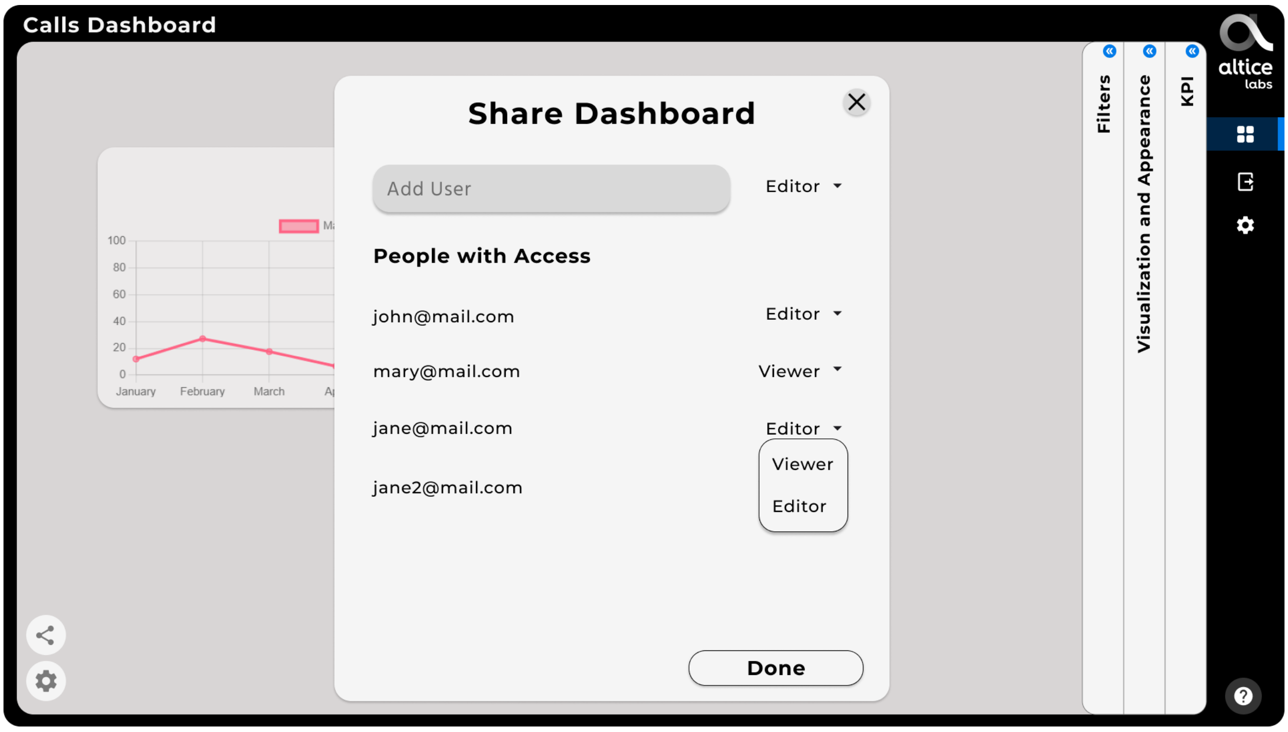


Figure 4.14: Mockup of the Widget on the Dashboard

Sharing a Dashboard

Users have the ability to share their dashboard with other users and manage the roles of previously shared individuals. Figure 4.15 shows the interface of this menu which draws significant inspiration from Google Drive's sharing method, providing a familiar and intuitive experience. When initiating the sharing process, a popup appears, dimming the rest of the screen to bring focus to the task at hand. Once the user has shared with a new user or finished updating existing sharing settings, the popup closes, allowing the user to seamlessly continue their workflow. This approach ensures a user-friendly and efficient sharing experience within the application, promoting collaboration and accessibility.



Share Dashboard

Add User — Editor

People with Access

john@mail.com	Editor
mary@mail.com	Viewer
jane@mail.com	Editor
jane2@mail.com	

Viewer
Editor

Figure 4.15: Mockup of Sharing the Dashboard

Viewer Mode

When a user is granted viewer access to a dashboard, certain actions such as creating, editing, resizing, or moving widgets are restricted. However, they retain the ability to apply filters to the widgets, enabling them to visualize the data with different filtering options. To indicate that the dashboard is in viewer mode, a label is added next to the title, providing clear visibility of the user's access level. This distinction helps users understand their permissions and ensures they are aware of their role within the dashboard environment. In Figure 4.16, all of the aforementioned remarks and functionalities can be observed visually.

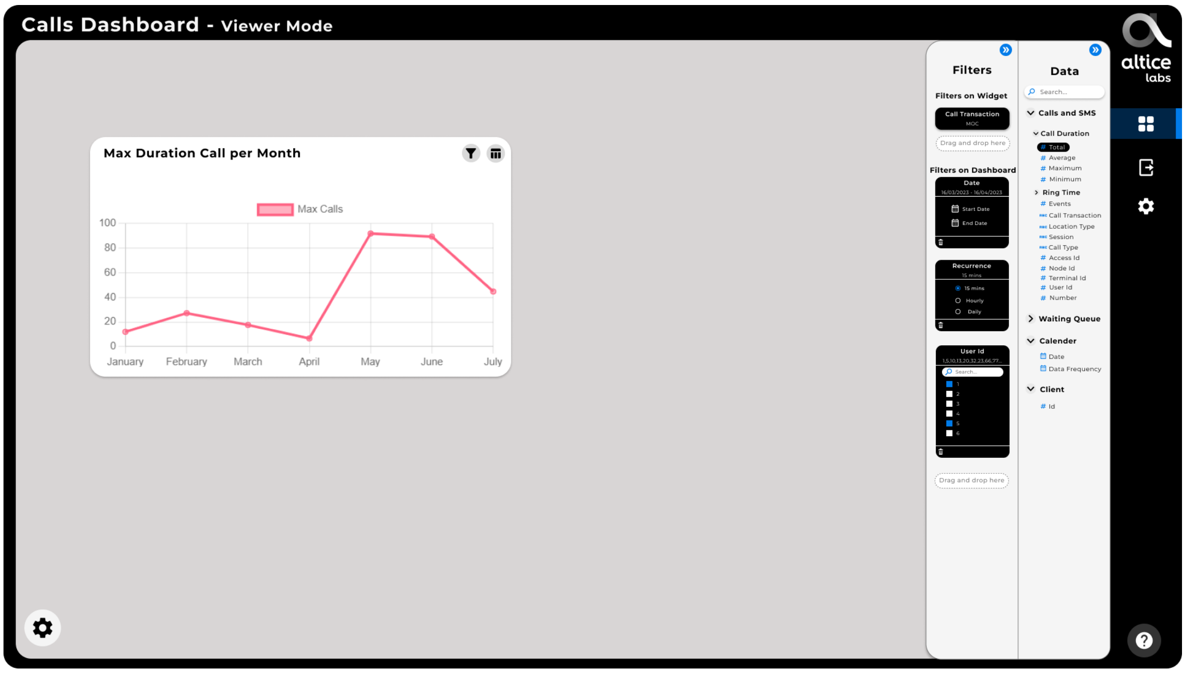
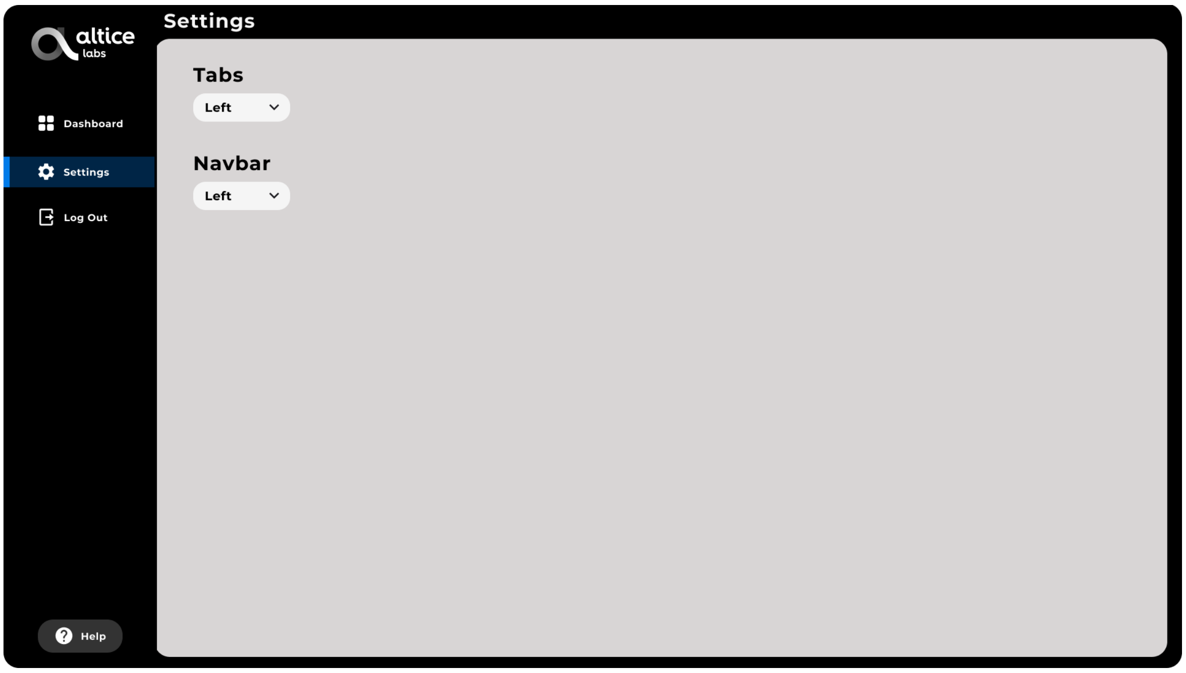


Figure 4.16: Mockup of the Dashboard on Viewer Mode

Settings

Figure 4.17 illustrates the settings menu, which currently allows users to change the display side of the tabs and sidebar. By separating the settings menu onto a different page, future development can accommodate additional options as needed.



Settings

Tabs

Left

Navbar

Left

Figure 4.17: Mockup of the Settings Page

4.7 Database Conceptual Diagram

This section provides a presentation and explanation of the database conceptual diagram presented in figure 4.18.

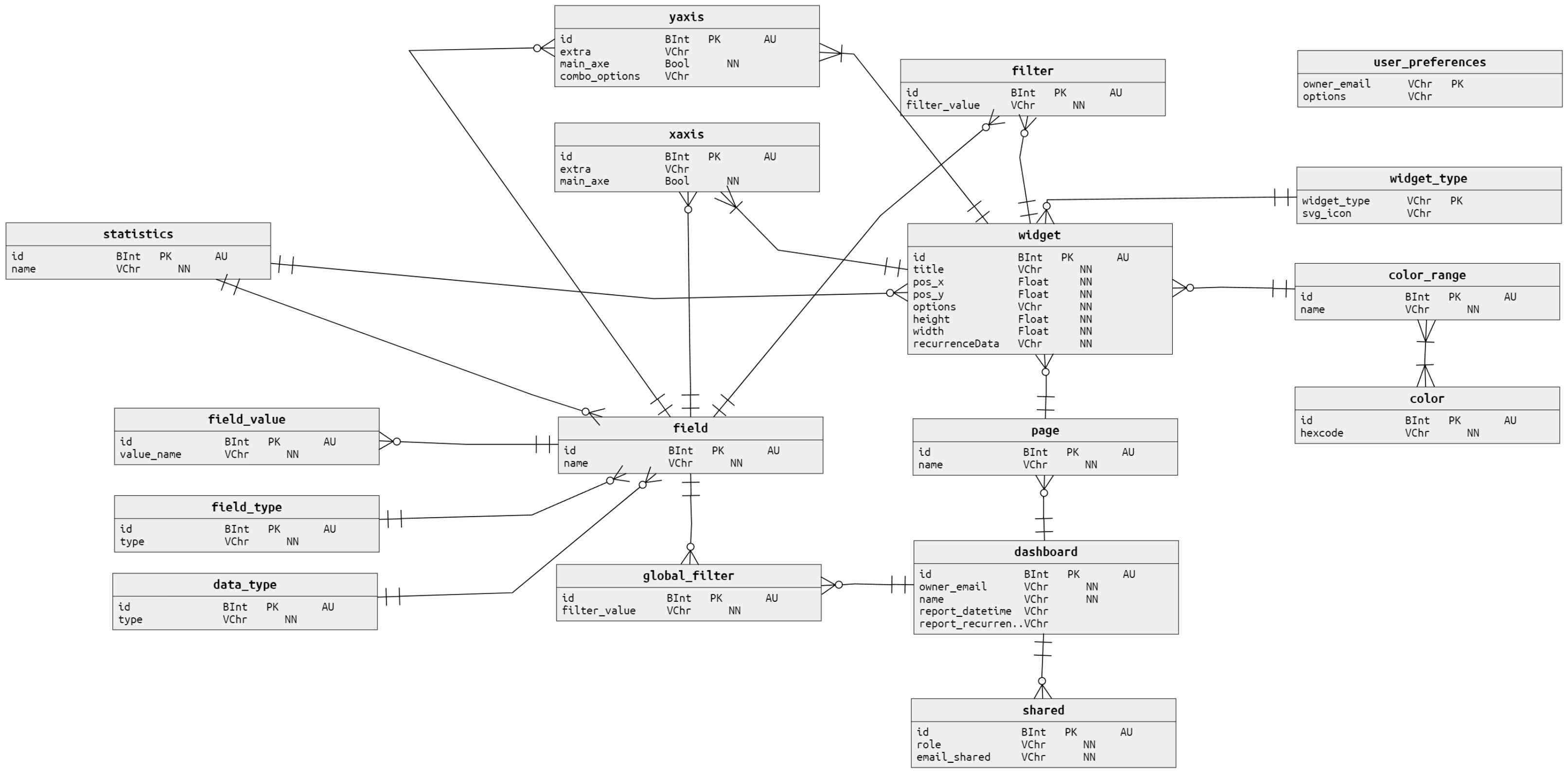


Figure 4.18: Database conceptual diagram

The database is divided into two sections: the statistics part and the dashboard part. The statistics part includes entities such as statistics, field, field_value, field_type, and data_type. The dashboard part comprises all the remaining entities.

In the statistics entities, data is pre-filled before deployment, and users only have read access to these tables. The statistic entity stores the names of KPIs, with each statistic having associated fields. These fields have types indicating whether they are metrics or dimensions, data types (such as string, number, or timestamp), and some fields may have predetermined values that are also filled.

A dashboard is associated with an owner, a name, and includes additional attributes like report datetime and recurrence, which will be utilized for future development of emailing reports. Dashboards can also be shared with other users. As mentioned earlier, a dashboard can have multiple pages, with each page containing widgets. The global_filter entity stores the filters applied to the dashboard, consisting of a field and the user-selected value.

A widget encompasses attributes such as chart type, color range, position, size, title, and data recurrence. Each widget can have filters applied to it. The xaxis and yaxis entities store the corresponding fields for each axis within the widget. Both xaxis and yaxis entities have an additional field to store extra information about the axis. In cases where multiple fields exist on the same axis, they indicate whether a particular field is the main one or not. The yaxis entity also includes a combo_options attribute to store information when the widget is a combination of other types, such as the line-column chart.

This page is intentionally left blank.

Chapter 5

Implementation

This chapter offers an overview of the development process and technical aspects involved in bringing the application to life. It covers the methodology employed for API and frontend development, the integration of these components, version control practices, and deployment procedures. Through an in-depth exploration of the implementation process, this chapter aims to provide a comprehensive understanding of the technical intricacies, decision-making processes, and challenges encountered during the development of the full-stack application.

5.1 Methodology

The development of the product followed an Iterative and Incremental Development methodology, as described in Chapter 2. Figure 5.1 illustrates how this methodology works.

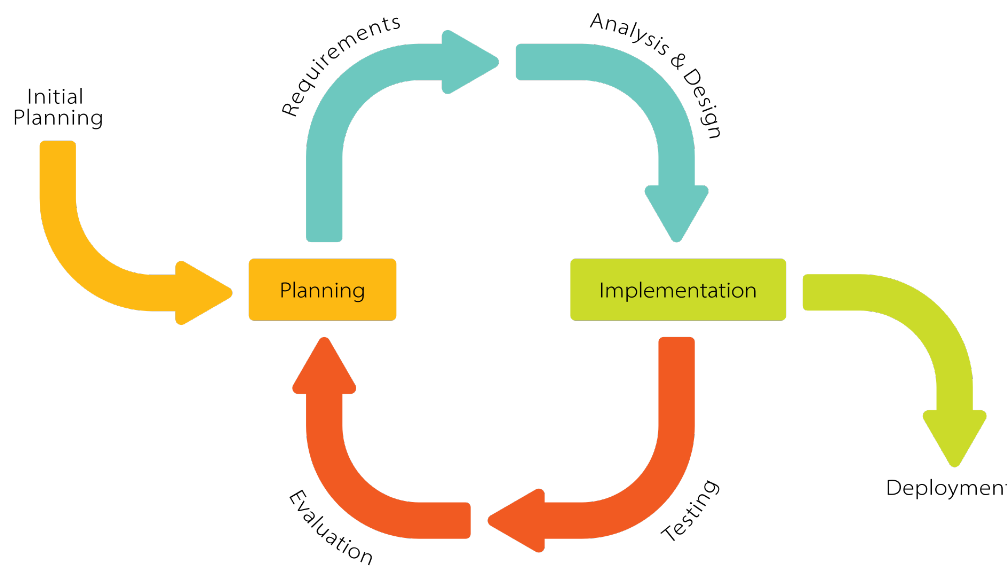


Figure 5.1: Iterative and incremental development

During the initial planning phase, the functional and non-functional requirements were defined and prioritized. Each level of prioritization marked the beginning of an iterative cycle.

For each cycle, the planning stage involved making changes to the architectural drivers. In the first cycle, an API schema was designed, specifying the names, request bodies, and response formats of each endpoint. This schema was revised and updated in subsequent cycles to accommodate new functionalities. Mockups also underwent changes in each cycle to reflect the evolving features. Additionally, the database structure was modified to incorporate new fields and entities required for these functionalities.

Following the planning phase and architectural adjustments, the implementation process began. API development was tackled first, followed by frontend development, and finally the integration of both components. This approach allowed for focused attention on one aspect at a time while ensuring sufficient time was allocated to develop each functionality.

Once the implementation phase was complete, the testing phase commenced. Tests were conducted for both the API and the frontend, with any discovered bugs promptly addressed before the start of a new cycle. Code reviews were performed by other team members to evaluate the quality of the code.

Throughout the development process, bi-weekly team meetings were held to present and explain the work accomplished, as well as to gather feedback from team members. These meetings served as an opportunity to discuss progress, address any concerns, and ensure alignment within the team.

The chosen methodology provided enhanced organization and focus by breaking down the development process into smaller tasks with clear deadlines. This approach facilitated a better understanding of task progress and identified any potential delays in the development timeline. Furthermore, the methodology fostered a culture of continuous improvement across various aspects, including development, planning, and architecture. This iterative approach allowed for ongoing refinement and optimization, resulting in a more efficient and effective development process.

In order to effectively track and manage the issues and tasks related to each development increment, Jira was used. The team manager was responsible for creating and assigning the relevant issues in Jira. These issues encompassed the different requirements, enhancements, and bug fixes associated with each development increment.

Fisheye, a code review tool, was utilized to facilitate code reviews and enhance collaboration among team members. Fisheye provides a platform for team members to review and provide feedback on the codebase, ensuring code quality, identifying potential issues, and promoting knowledge sharing within the team. Team members could easily access the code repository and view the changes made in each commit or branch. They had the ability to leave comments, suggest improvements, and address any concerns directly on the code. This interactive feedback process helped in identifying areas for improvement, promoting code consistency, and ensuring adherence to best practices.

5.2 Version Control

At AlticeLabs, the chosen VCS tool is Subversion (SVN), developed by Apache.

The SVN repository is integrated with Fisheye and Jira. This integration allows team members to easily track which code versions were associated with specific issues and provides an efficient way to review code changes using Fisheye. This integration enhances the overall development workflow and promotes code quality through code reviews.

Overall, the use of SVN, along with its integration with Fisheye and Jira, enhances the development process at AlticeLabs by providing version control, collaboration capabilities, and code review functionality.

5.3 Deployment

Deployment is a crucial step in the software development lifecycle, as it involves transitioning a web application from the development environment to a production environment where it can be accessed by end-users through their web browsers. This process entails packaging the application and configuring it to run on a production server.

For this project, the deployment was accomplished using Nginx, a popular web server software that can efficiently handle high traffic and serve web applications.

However, the deployment process deviated from the initial plan of continuous deployments after each increment. Due to compatibility issues with the Node version used in both projects, the deployment had to be postponed until the end of the timeline until a team member updated the machine to support the required Node version.

To deploy the application, the build command was executed in both projects to generate the production-ready code. The resulting build artifacts were then transferred to the deployment machine using FTP, following the specific procedures set by AlticeLabs.

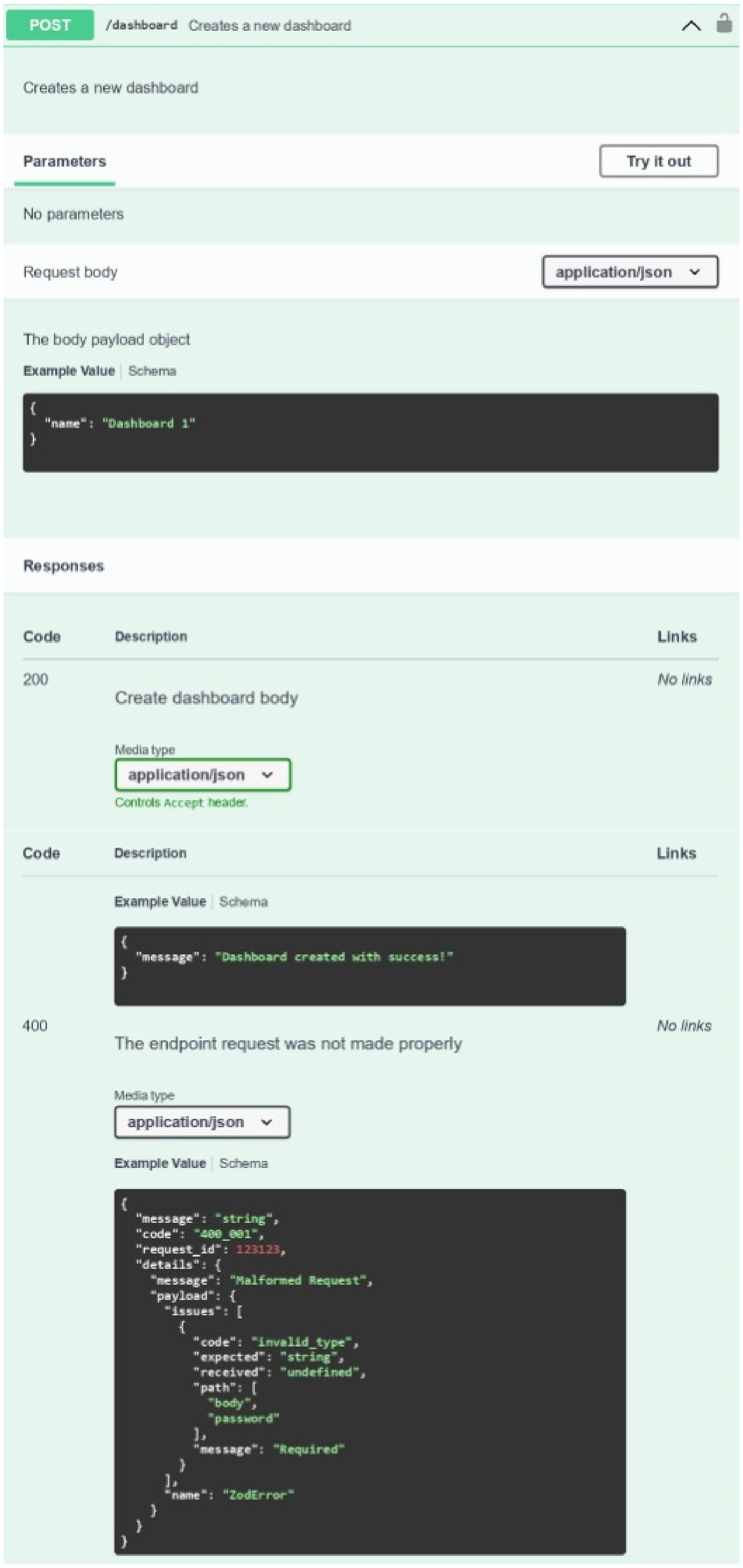
To enhance flexibility and maintainability, the application utilized the dotenv package to store configuration variables in environment files. This approach allowed for easy modification of various settings, such as the database URL, frontend and API URLs for CORS, without the need to recompile the code with each change.

5.4 API Development

This section provides an overview of the planning and development process involved in creating the backend API. By delving into the details, we can gain a comprehensive understanding of how the API was developed.

5.4.1 API Schema

The development of the API embraced an API-first approach, where the API's contract, outlining its expected behavior, was defined prior to implementing its actual functionality. This methodology ensured that each endpoint followed a coherent and logical pattern, and provided an opportunity to review and validate each endpoint before development commenced. Leveraging an iterative development process, this API-first approach facilitated incremental enhancements and seamless iteration across different components without disrupting the overall system architecture. It also fostered improved collaboration and facilitated integration, as team members could readily review the API design. The comprehensive schema encompassed detailed information for each endpoint, including available responses, request bodies, parameters, and queries. Figure 5.2 illustrates an example of one endpoint within the schema.



Creates a new dashboard

No parameters

The body payload object

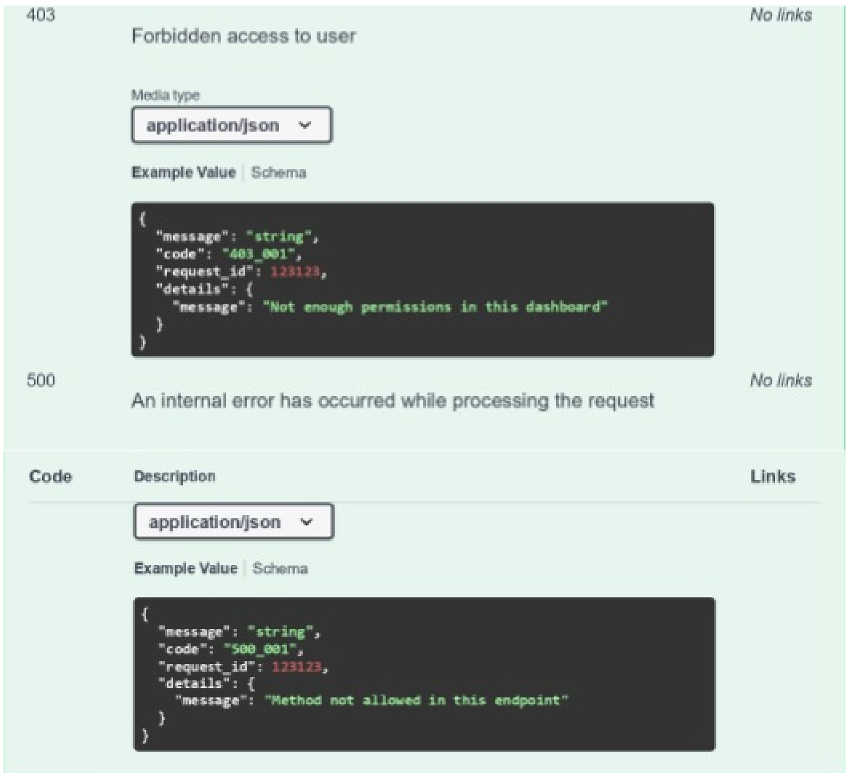
```
{
  "name": "Dashboard 1"
}
```

Code	Description	Links
200	Create dashboard body	No links

```
{
  "message": "Dashboard created with success!"
}
```

Code	Description	Links
400	The endpoint request was not made properly	No links

```
{
  "message": "string",
  "code": "400_001",
  "request_id": 123123,
  "details": {
    "message": "Malformed Request",
    "payload": {
      "issues": [
        {
          "code": "invalid_type",
          "expected": "string",
          "received": "undefined",
          "path": [
            "body",
            "password"
          ],
          "message": "Required"
        }
      ],
      "name": "ZodError"
    }
  }
}
```



403 Forbidden access to user — No links

Media type: application/json

Example Value | Schema

```
{
  "message": "string",
  "code": "403_001",
  "request_id": 123123,
  "details": {
    "message": "Not enough permissions in this dashboard"
  }
}
```

500 An internal error has occurred while processing the request — No links

Code	Description	Links

application/json

Example Value | Schema

```
{
  "message": "string",
  "code": "500_001",
  "request_id": 123123,
  "details": {
    "message": "Method not allowed in this endpoint"
  }
}
```

Figure 5.2: Example of endpoint documentation, screenshoted from OpenAPI

5.4.2 Authentication

To authenticate users, the API utilizes JWT. JWTs are encoded JSON data with a cryptographic signature appended at the end. A JWT comprises three parts: the header, the payload, and the signature. In the payload, user information such as the user's ID and email is included.

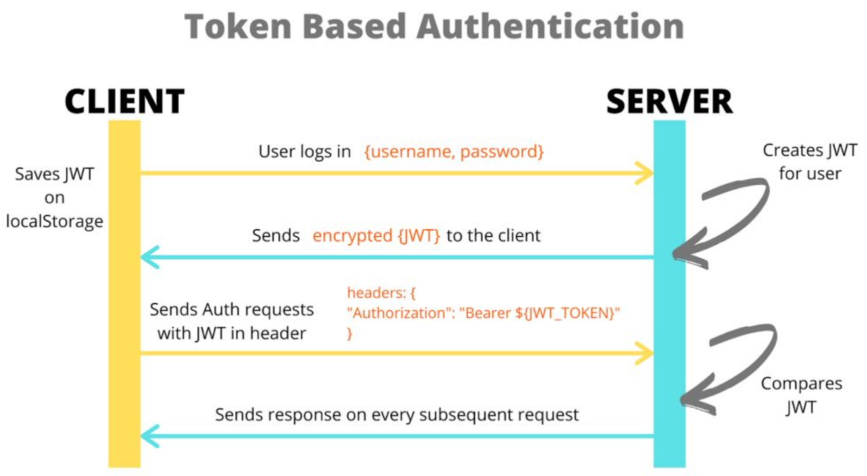

Token Based Authentication
CLIENT
SERVER
Saves JWT on localStorage
User logs in {username, password}
Creates JWT for user
Sends encrypted {JWT} to the client
Sends Auth requests with JWT in header
headers: { "Authorization": "Bearer ${JWT_TOKEN}" }
Compares JWT
Sends response on every subsequent request

Figure 5.3: JWT authentication diagram, from [30]

Figure 5.3 illustrates the authentication process with JWT. Upon successful login, the API generates and returns a JWT, which the browser stores in local storage or cookies. Subsequently, when the user makes an API call, the JWT is added to the request header, and the system verifies its validity. Each endpoint then verifies if the user making the API request corresponds to the user specified in the JWT payload. This prevents unauthorized users from interfering with or accessing other users' information.

To enhance the security of the JWT, a timeout of 15 minutes has been implemented. After this period, the JWT becomes invalid. However, alongside the JWT, a refresh token is sent to the browser and stored. When the JWT expires, an automated API call is triggered to a designated endpoint responsible for refreshing the JWT. Along with the refreshed JWT, a new refresh token is issued.

The API leverages JWT-based authentication along with these supplementary security measures to establish robust user authentication and safeguard sensitive user data against unauthorized access. Additionally, JWT enables a stateless server architecture, eliminating the need to maintain user-related state information and enhancing scalability. By adopting JWT, a token-based authentication approach, it lays the foundation for future development of a single sign-on (SSO) functionality. This enables users to authenticate once and gain access to multiple applications or services seamlessly. Thus, JWT-based authentication not only ensures security and scalability but also offers potential for convenient and streamlined user experience.

5.4.3 Endpoints

The API endpoints at AlticeLabs are designed and implemented using best practices to ensure consistency, standardization, ease of use, and seamless integration. Adhering to these practices enables scalability, extensibility, and optimal performance. It also enhances security, access control, and collaboration among developers. By following best practices, the APIs become future-proof, compatible, and provide an excellent developer experience [68].

Some of these best practices include [19]:

- **Using JSON for data transmission**: JSON enhances data readability, enjoys wide support across applications, is easily parsed by various technologies, and is more cache-friendly.
- **Utilizing nouns instead of verbs in endpoint naming**: RESTful Uniform Resource Identifier (URI) refer to a resource, a thing, therefore it should be a noun that is capable of having properties;
- **Properly utilizing HTTP status codes**: Adhering to the correct meaning of HTTP status codes aids developers in handling errors or unexpected success scenarios.

- **Using nested endpoints to indicate relationships**: When endpoints are interconnected, nesting them logically reflects their relationship, making it easier to understand and navigate the API structure.
- **Implementing filters and pagination for data retrieval**: To improve processing times, utilizing filters and pagination allows developers to retrieve specific data, avoiding unnecessary overhead and speeding up response times.
- **Providing accurate and comprehensive API documentation**: Documentation should encompass all endpoints, their specifications, and relevant status messages.

With this in mind, the endpoints all follow the same naming convention:

```
api/<collection_name>/<endpoint_name>
```

The endpoints developed are:

- General: **/**
  - **GET healthcheck/**: This endpoint indicates the health of the API, returning a 200 if everything is ok or a specific error code to indicate what's wrong.
- Authentication: **auth/**
  - **POST login/**: Checks if the user exists and, if successful, returns the JWT and refresh token.
  - **POST refresh/**: When called, if the refresh token has not expired, will return a new set of JWT and refresh token.
  - **GET token-up-to-date/**: Checks if the token is up to date or has expired.
- User: **user/**
  - **GET preferences/**: Gets all of the user's UI preferences.
  - **PUT preferences/**: Updates the user's preferences.
- User: **color/**
  - **GET /**: Gets all of the color ranges for widget creation available.
- Dashboard: **dashboard/**
  - **GET /**: Gets all of the user's dashboards (created by and shared with).
  - **POST /**: Creates a new dashboard on the user's account.
  - **PUT /{dashboardId}**: Updates the dashboard with the ID `dashboardId`.
  - **DELETE /{dashboardId}**: Deletes the dashboard with ID `dashboardId`.

  - **GET /{dashboardId}**: Gets all the information about a specific dashboard.
  - **GET /{dashboardId}/page/{pageId}**: Gets all the widgets created on the specific dashboard page.
  - **GET /{dashboardId}/share**: Gets all the shared users of the dashboard.
  - **POST /{dashboardId}/share**: Shares the dashboard with a new user.
  - **PUT /{dashboardId}/share**: Updates the user's role on the dashboard.
  - **DELETE /{dashboardId}/share**: Unshares the dashboard with a new user.
  - Widget: **dashboard/page/{pageId}/widget/**
    * **POST /**: Creates a new widget on the specific page of the dashboard.
    * **PUT /{widgetId}**: Updates the widget configurations.
    * **DELETE /{widgetId}**: Deletes the widget.

- Statistics: **statistics/**
  - **GET /**: Gets the list of all KPIs and their metrics and values.
- Analytics: **analytics/**
  - **POST /**: Transforms widget configuration into data that can be given to the chart builder.
  - **GET /labels**: Lists all of the KPIs labels to use on the frontend application for translation.
  - **GET /filter-values**: Gets a list of all available values to filter by.

In order to ensure consistent responses across all endpoints, the following response codes were defined:

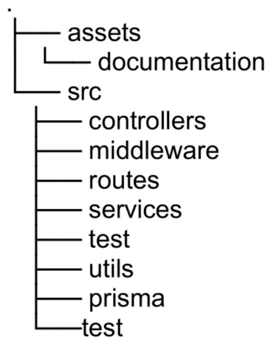
- Ok-200: This response indicates that the endpoint has successfully processed the request. Along with this response, a response body is sent, which may vary depending on the specific endpoint;
- Bad Request-400: If the provided request body does not match the expected format or contains invalid data, a 400 error code is returned to the user;
- Unauthorized-401: If the user attempting to access the resource lacks the necessary permissions or fails to provide valid authentication credentials, a 401 error code is returned;
- Forbidden-403: If the user has been explicitly excluded from accessing the resource or lacks sufficient permissions, a 403 error code is sent as a response;
- URL not Found-404: When a request is made to a non-existing URL, a 404 error code is returned, indicating that the requested resource could not be found;

- ID not Found-404: If the user includes an ID in the request body or as a parameter that does not correspond to an existing resource, a 404 error code is returned along with a message indicating which field contains the non-existing ID;
- Method Not Allowed-405: When the user makes a request using an HTTP method that is not supported or allowed for the specific endpoint, a 405 error code is returned;
- Internal Server Error-500: In the event of an unexpected server failure or error, a 500 error code is returned to indicate that there are problems on the server side.

By defining these standardized response codes, users interacting with the API can expect consistent and meaningful feedback based on the outcome of their requests.

5.4.4 API Framework and Organization

The API development was undertaken using Express.js as the chosen framework. To make the code more mantainable the structure was made taking into account similar projects at AlticeLabs and Express.js best practices [44]. Figure 5.4 displays the tree of the project.



```
.
├── assets
│   └── documentation
└── src
    ├── controllers
    ├── middleware
    ├── routes
    ├── services
    ├── test
    ├── utils
    ├── prisma
    └──test
```

Figure 5.4: Folder Tree of the API Project

The project consists of two main folders: "src" and "assets" The "src" folder houses all the project files, while the "assets" folder contains the OpenAPI documentation file.

The "prisma" folder contains the entity models for the database. Prisma, as an Object-Relational Mapping (ORM) tool, acts as a bridge between the application and the database. It maps the entities onto objects, facilitating seamless interaction and providing an abstraction layer for SQL queries. Prisma simplifies the learning curve by allowing developers to use easy-to-understand and code

CRUD (Create, Read, Update, Delete) methods. Additionally, Prisma helps ensure code maintainability by providing accurate error messages when there are updates to the database schema that could potentially break the existing code.

Within the "src" folder, the "utils" directory contains methods and functions needed across the application. The "config.ts" file loads environment configurations, such as the database connection and frontend URL for CORS (Cross-Origin Resource Sharing). Logging within the project is implemented using Pino, allowing for message logging throughout. The logging behavior varies based on the environment, providing different information depending on whether the project is in development or deployment. The "schemas.ts" and "types.ts" files define the request and response bodies for the endpoints, while "responses.ts" contains boilerplate code to ensure consistency across all responses.

The "controller" folder contains the logical implementation of the endpoints, taking in requests and transforming them into responses.

The "routes" folder houses the logical setup for routing.

The "services" folder contains the business logic, such as the authentication mechanism.

By organizing the project's files into these distinct folders, the codebase is structured in a way that separates concerns, enhances code organization, and improves maintainability.

5.5 Front End

This section provides an overview of how the front-end code is organized, the different components that make up the user interface, and the specific pages that define the user experience. By understanding the project structure, components, and pages, it becomes easier to grasp the overall architecture and functionality of the front-end system.

5.5.1 Project Structure

Figure 5.5 shows the structure of the front-end project, which follows a consistent pattern used in other AlticeLabs front-end projects. The goal was to make the code accessible for future work and maintainable by other team members

```
.
├── public
│   └── locales
│       ├── en
│       └── pt
└── src
    ├── assets
    │   ├── css
    │   └── images
    ├── components
    ├── config
    ├── interfaces
    ├── routes
    ├── layouts
    ├── middleware
    ├── pages
    ├── routes
    ├── styleguide
    ├── tests
    └── util
```

Figure 5.5: Folder Tree of the Front End Project

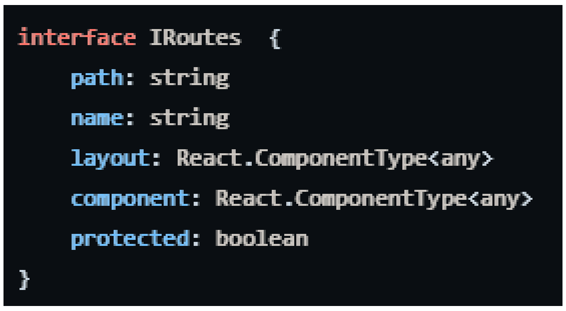
In the "public" folder, the translations JSON files are located. These files are used for internationalization (i18n), allowing for easy translation of labels and texts into different languages in the future.

The "src" folder contains all the code for the project. Within the "src" folder, the "assets" folder stores images and ".scss" files. In this project, Tailwind and Sass are used for styling customization. Tailwind provides pre-defined classes for quick and consistent styling, while Sass introduces variables and nesting for enhanced organization and reusability of styles. This combination allows for efficient development, cleaner code, improved productivity, and easier maintenance.

The "util" folder includes helper functions that support various aspects of the project. For example, the "api.ts" file handles HTTP requests and provides methods for CRUD operations. The "cookie.ts" file utilizes the "universal-cookie" package to store and retrieve cookies from the browser, specifically storing the JWT and refresh token. The "windowSize.ts" file determines the current window size of the user's browser for responsive UI design. The "i18n.ts" file connects the translations from the "public" folder to the project.

In the "config" folder, you can find the configuration files. The "config.ts" file reads configurations from the environment and makes them accessible to other files. The "logger.ts" file enables logging of information and messages, facilitating the debugging process.

The "route" folder consists of files that map the web application routes. By organizing routes in a separate folder, it becomes easier to add new endpoints or modify existing ones. Figure 5.6 illustrates how the route mapping works, with each route having a path, name (used for translation), layout (container for components), and an optional middleware function for user authentication.



```
interface IRoutes  {
    path: string
    name: string
    layout: React.ComponentType<any>
    component: React.ComponentType<any>
    protected: boolean
}
```

Figure 5.6: Interface of Route Mapping

The "middleware" folder contains files with functions that are called before rendering a page. For example, the "ProtectedRoute.tsx" function verifies if the user is authenticated before redirecting to the appropriate page.

The "layouts" folder contains the layout components used in the project. Layouts act as containers for components that are shared across multiple pages, ensuring consistent rendering. Each layout receives a page component as a child and renders it accordingly.

The "Components" folder includes reusable building blocks that can be used in different pages, making it easier to manage the interface and update components consistently.

The "pages" folder contains all the web application pages. These pages are built using components and include their own internal logic. Each page is responsible for rendering a specific part of the application based on the defined routing. The next subsection will present and explain all the pages in detail.

The "styleguide" folder contains the configuration for the documentation tool called "Styleguidist." This tool plays a crucial role in documenting all the components used in the project. It provides examples of component usage and documents the available props for each component. By leveraging Styleguidist, future developers can quickly understand how to use components and gain insights into their functionality. This approach enhances maintainability by promoting a clear understanding of the components' purpose and usage.

5.5.2 Components

This subsection presents all the components along with their props, offering insight into the development process. It serves as a comprehensive overview of the components used in the project, allowing for a deeper understanding of their purpose and usage. All of this examples are taken from the documentation done to the components, that can be found in appendix C.

To evaluate the reusability of components, a metric was employed. The number of reusable components was divided by the total number of components and then multiplied by 100. This calculation yielded a reusability rate of 84.6%, indicating that the non-functional requirement for component reusability has been met.

Button

By creating a button component, we can ensure consistent styling and provide a seamless user experience across the web application. This versatile component caters to various button functionalities based on the props passed to it. Whether it's a normal button, a form submit button, or a return button, the button component can adapt accordingly. Additionally, the component supports a disabled state, ensuring it covers all the necessary use cases for a button.

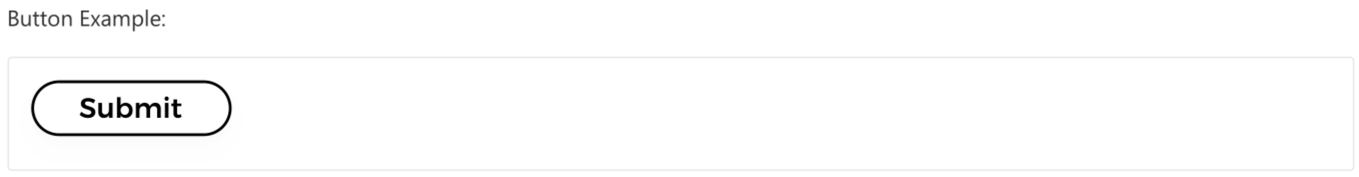


Figure 5.7: Button Component Example

PopUp

Figure 5.20 illustrates the properties and provides an example of a popup component. The popup component is highly versatile and can be easily adapted to various use cases by utilizing the "children" prop, which allows for dynamic content rendering. Although not explicitly demonstrated in this example, when the popup is displayed, the background is dimmed to draw attention to the popup itself. Additionally, the popup component offers customization options for defining the behavior when the popup is closed, providing flexibility for different scenarios.

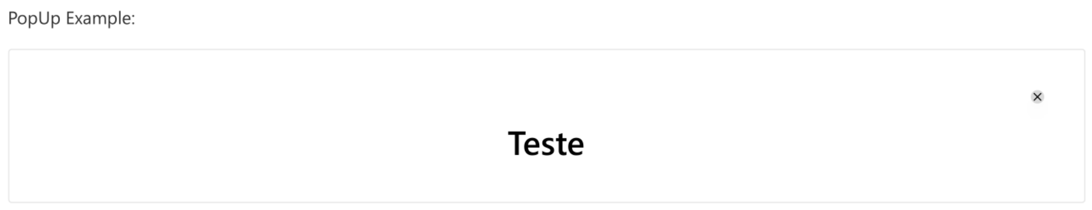


Figure 5.8: PopUp Component Example

FAB

The floating action button (FAB) serves as a convenient means of accessing primary or frequently used actions within the application without occupying excessive space. When the FAB is hovered, additional buttons, referred to as action buttons, appear. The FAB adjusts its position based on the user's chosen UI layout. Figure 5.9 presents an example of the FAB component.

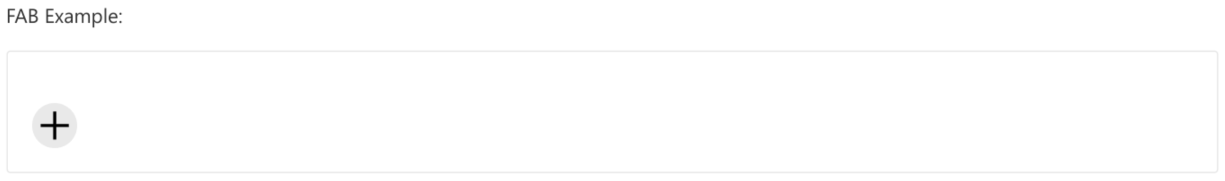


Figure 5.9: FAB Component Props Example

Side Bar

The sidebar component, depicted in Figure 5.10, serves as a navigation hub for users to access different pages within the application. When the sidebar is hovered over, it expands to reveal its contents, allowing users to easily navigate between pages. Conversely, when the mouse pointer is no longer hovering over the sidebar, it automatically collapses, freeing up space within the application interface. This behavior optimizes the available screen real estate, providing users with a larger workspace while still offering convenient access to the application's pages.

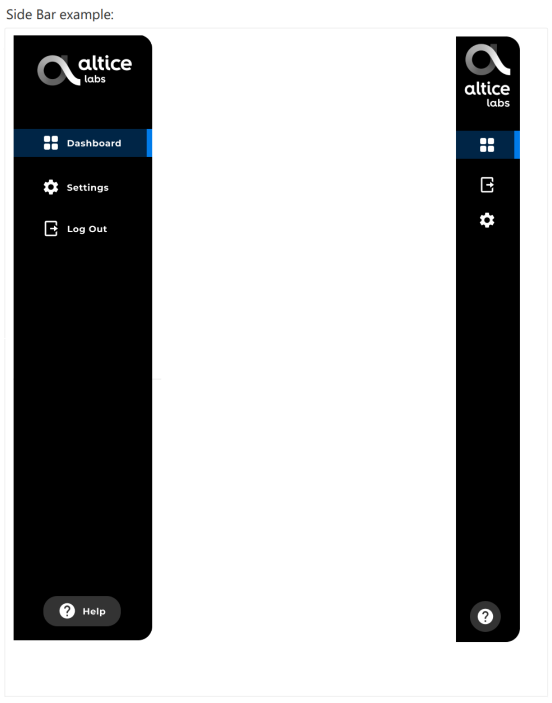


Figure 5.10: Side Bar Component Example

Toolbar

The toolbar component, showcased in Figure 5.23, offers a versatile and adaptable toolbar feature. It can be toggled open or closed, allowing users to control its visibility as needed. The toolbar component is designed to support multiple instances within the same parent page, providing the flexibility to incorporate multiple toolbars throughout the application. By accepting React "children" components, it enables seamless reuse and customization, empowering developers to add various elements and functionalities to the toolbar. This modularity and configurability make the toolbar component a valuable asset for enhancing the user experience and providing convenient access to essential actions and features.

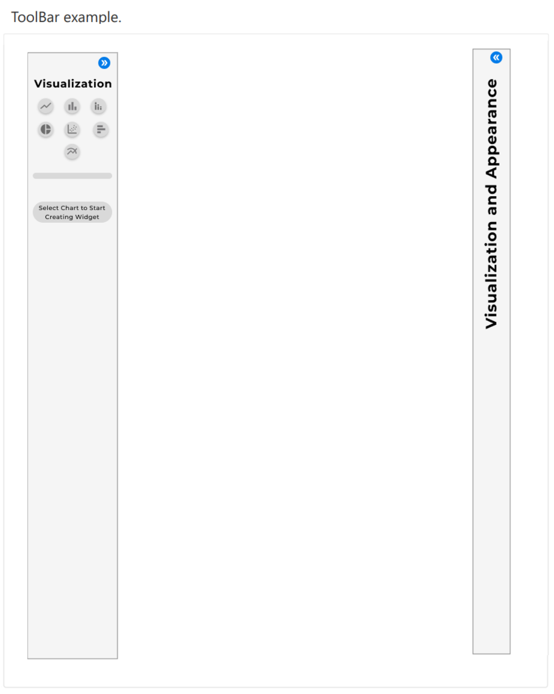

ToolBar example.
Visualization
Select Chart to Start Creating Widget
Visualization and Appearance

Figure 5.11: Toolbar Component Props Example

Dashboard Card

The dashboard card component, illustrated in Figure 5.12, empowers users to interact with their dashboards efficiently. It offers essential functionality such as opening, editing, and deleting dashboards. The card prominently displays the name of the dashboard, providing users with clear identification and easy navigation. Additionally, the dashboard card includes relevant information about the user's role or permissions associated with the dashboard, ensuring proper context and facilitating appropriate actions. With its intuitive design and comprehensive features, the dashboard card component enhances the user experience and streamlines dashboard management within the application.

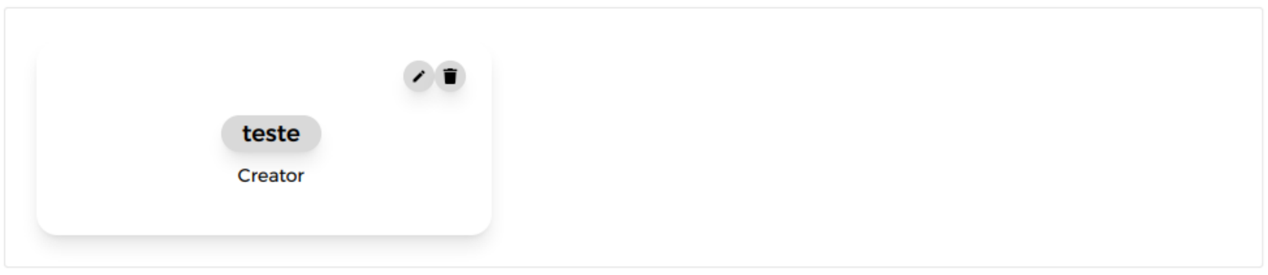


Figure 5.12: Dashboard Card Component Example

Widget Card

The widget card component, presented in Figure 5.13, serves as a versatile tool for users to interact with charts effectively. It offers a range of interactive features, including the ability to move, resize, and zoom the chart. Users can also interact with the chart by clicking on it to apply interactive filters or perform specific actions.

This component is designed to facilitate easy editing and deletion of the widget, providing users with control over their chart customization. By utilizing the React RnD library, the widget card component offers a seamless and intuitive user experience. The React RnD library is renowned for its capabilities in enabling draggable, resizable, and interactive components, making it an ideal choice for creating this widget card.

The integration of the React RnD library enhances the usability and flexibility of the widget card component. It empowers users to customize their charts effortlessly while maintaining a smooth and intuitive interface. The combination of the widget card component and the React RnD library ensures a seamless and efficient user experience when working with charts within the application.

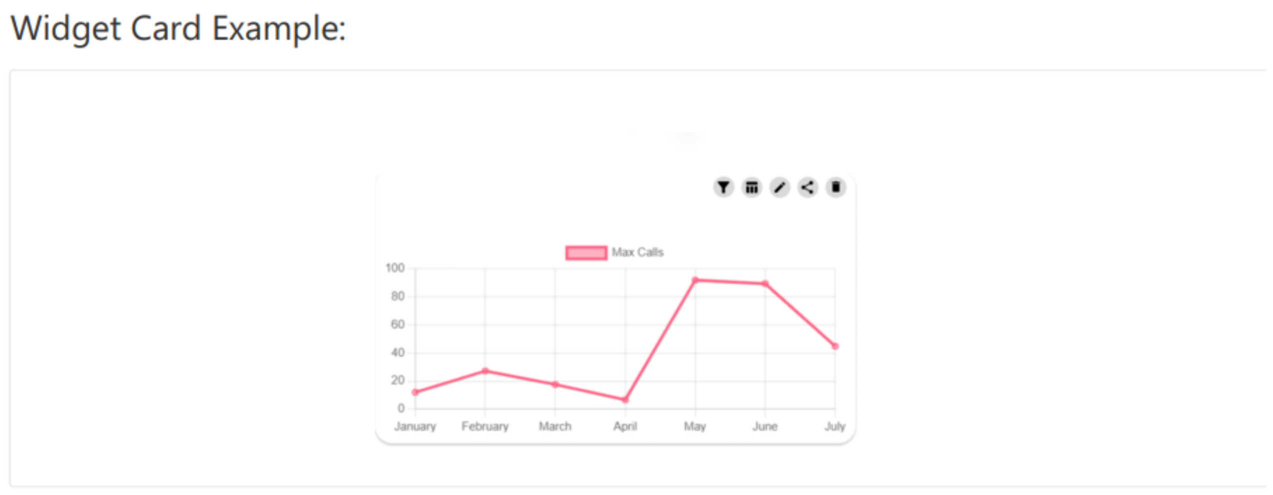


Figure 5.13: Widget Card Component Props and Example

Draggable and Dropzone

The draggable and dropzone components, displayed in figure 5.14 and figure 5.15, work hand in hand to provide essential functionality within the application. These components were developed using the React DnD library, which streamlines the process of implementing drag and drop functionality.

The draggable component allows users to select and drag desired metrics from a designated area. This functionality is particularly useful when creating a new widget, as users can easily drag and drop metrics onto the desired axis for visualization. By leveraging the React DnD library, the draggable component offers a seamless and intuitive drag experience.

On the other hand, the dropzone component serves as the target area where users can drop the draggable metrics. In the context of widget creation, the dropzone component enables users to add selected metrics to the chart's axis or apply them as filters. The React DnD library seamlessly handles the integration of the dropzone component, making it effortless for users to drop their metrics into the desired location.

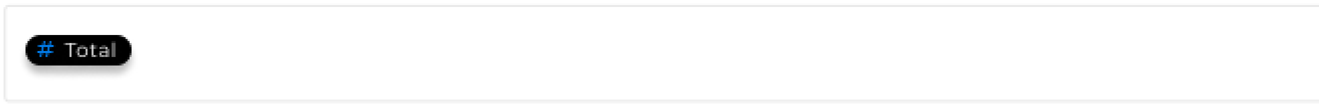


Figure 5.14: Draggable Component Example

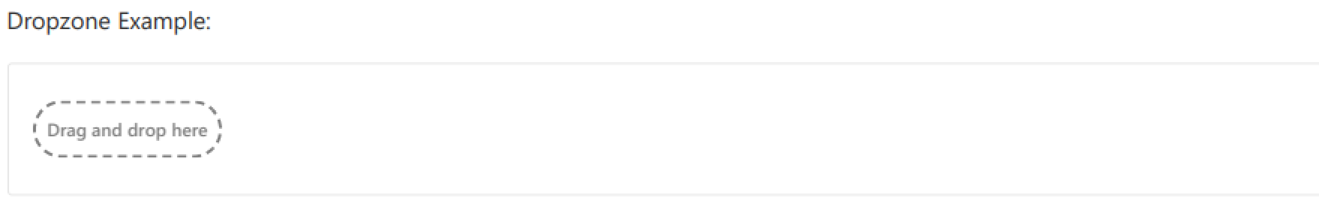


Figure 5.15: Dropzone Component Props Example

Dropdown

The dropdown component, as shown in Figure 5.16, provides a user-friendly way to present a list of elements. This component is highly customizable, allowing for the display of a large number of elements while providing a convenient scrollbar when the list exceeds a defined number of items.

With the dropdown component, users can easily access and select items from the list by expanding the dropdown menu. The component offers flexibility in terms of styling and behavior, enabling developers to tailor it to their specific needs.

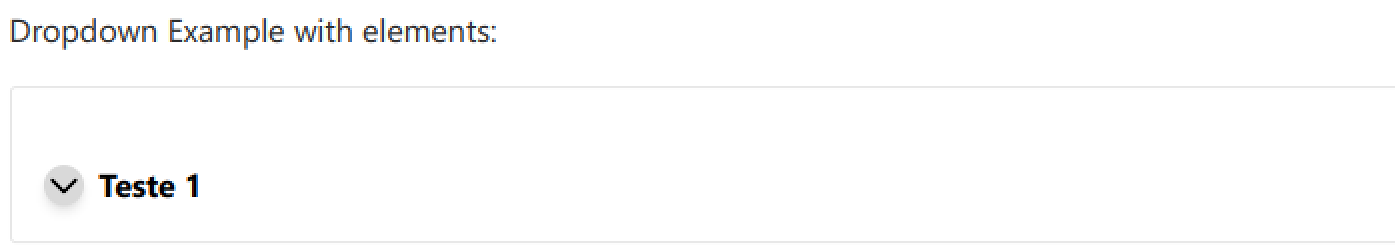


Figure 5.16: Dropdown Component Example

Filter Card

Filter cards are essential components that facilitate the creation and management of filters within an application. These components, as depicted in Figure 5.17, can be expanded or collapsed to optimize screen space. They offer built-in search functionality, enabling users to easily find specific values associated with the filter. Additionally, pagination is implemented to display only a specified number of elements at a time, preventing overwhelming the user with excessive information.

In contrast to the standard filter cards, the date and recurrence filters have their own dedicated components. Figure 5.18 showcases the date filter component, which allows users to edit the start and end dates by simply clicking on the corresponding text fields. This intuitive interface streamlines the process of selecting date ranges for filtering purposes. On the other hand, Figure 5.19 presents the available options for data collection recurrence. This specialized component offers a clear and concise representation of different recurrence options, enabling users to specify how data should be collected.

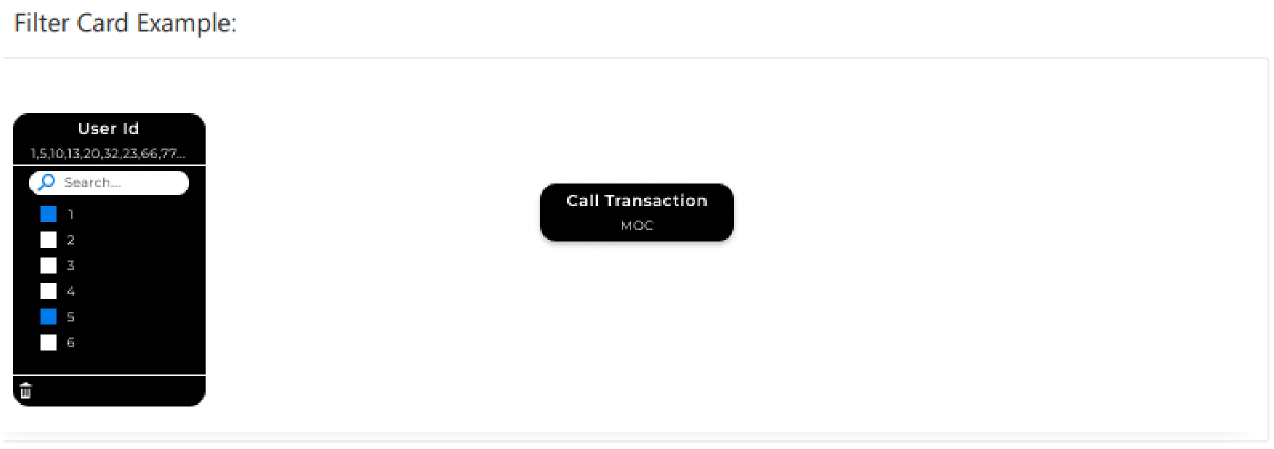
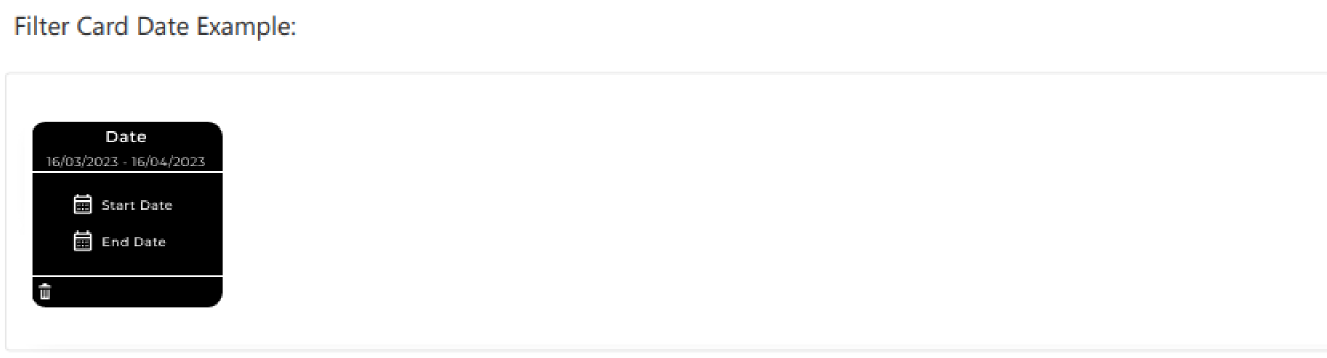
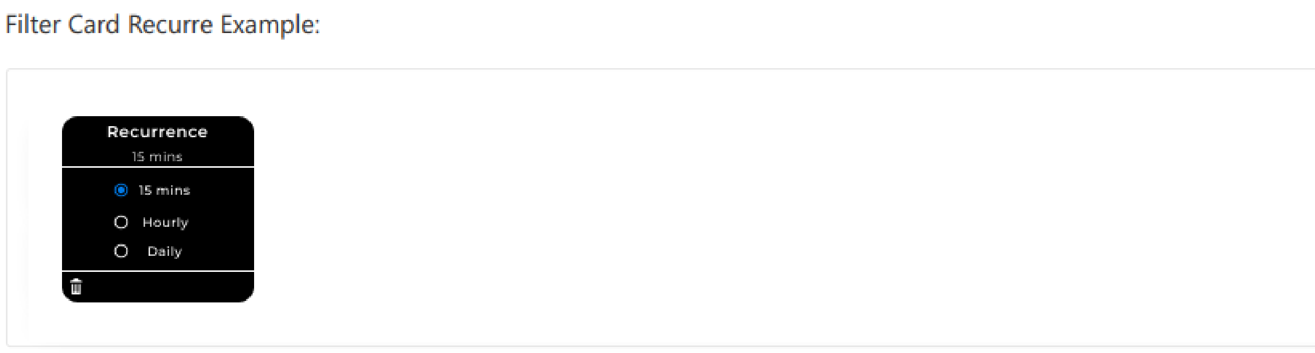


Figure 5.17: Filter Card Component Example


Filter Card Date Example:

Date
16/03/2023 - 16/04/2023
Start Date
End Date

Figure 5.18: Filter Card Date Example


Filter Card Recurre Example:

Recurrence
15 mins
15 mins
Hourly
Daily

Figure 5.19: Filter Card Recurrence Example

Palette Choice

The palette picker component functions similarly to a dropdown mechanism but with a specific focus on selecting color ranges. Instead of presenting a list of items, the palette picker provides users with a range of color options to choose from. This feature allows users to select the desired color range to apply to their widget.

By utilizing the palette picker, users can easily customize the visual appearance of their widgets by selecting color schemes that best suit their preferences or align with their branding requirements. This component streamlines the process of color selection and ensures consistency throughout the application by offering predefined color ranges for users to choose from.

The palette picker enhances the user experience by providing an intuitive and convenient way to incorporate visually appealing color schemes into their widgets, contributing to a more visually engaging and cohesive interface.

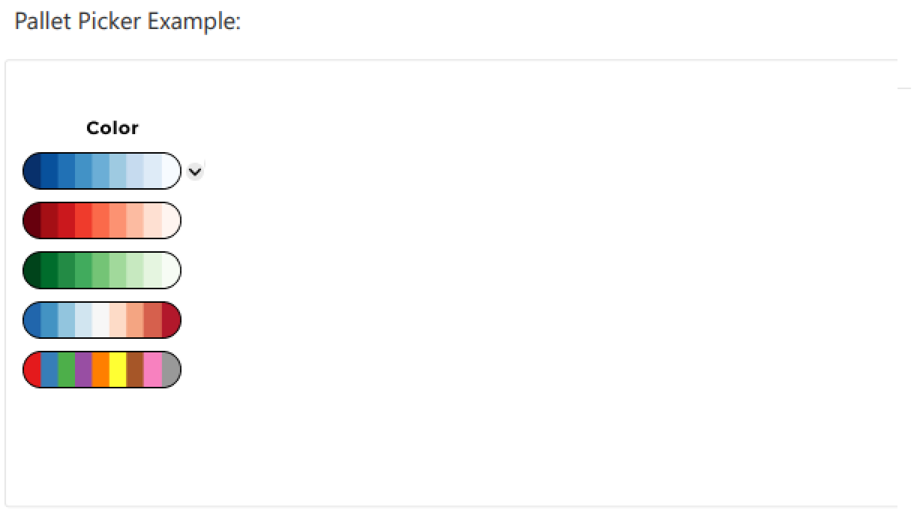


Figure 5.20: Button Component Props and Example

Charts

The Chart.js library played a crucial role in building the charts for this project. Known for its simplicity and maintainability, Chart.js provided the necessary tools for interactive chart creation and customization.

To prepare the data from the KPIs for charting, it needed to be transformed into a specific format that Chart.js could read. This transformation was facilitated by the "analytics/" endpoint, which returned the data in a JSON object format with "labels" and "datasets" keys. The "labels" represented the x-axis values, while the "datasets" contained the corresponding y-axis values. Each dataset within the "datasets" list had customizable properties, such as color, border color, and interpolation method for calculating click distances on the chart.

Chart.js also allowed for chart interactivity. While this functionality was not provided by default, it could be implemented through custom code. Two interactive features were incorporated: dataset hiding and global filtering. In charts with multiple datasets, users could hide all other datasets by clicking on either the dataset label or the specific dataset on the chart. This enabled users to focus on a particular dataset with a simple button click. Additionally, users could add global filters by clicking on specific values within the dataset on the chart.

Furthermore, plugins were utilized to enhance the charting experience. Zooming and panning functionality allowed users to focus on specific sections of the chart, enabling a closer examination of the desired data.

5.5.3 Pages

This chapter provides an overview of the various pages that make up the web application. As the functionality and user experience have been extensively discussed during the presentation of mockups and the implementation process, only a concise explanation of each page is given.

Main Page

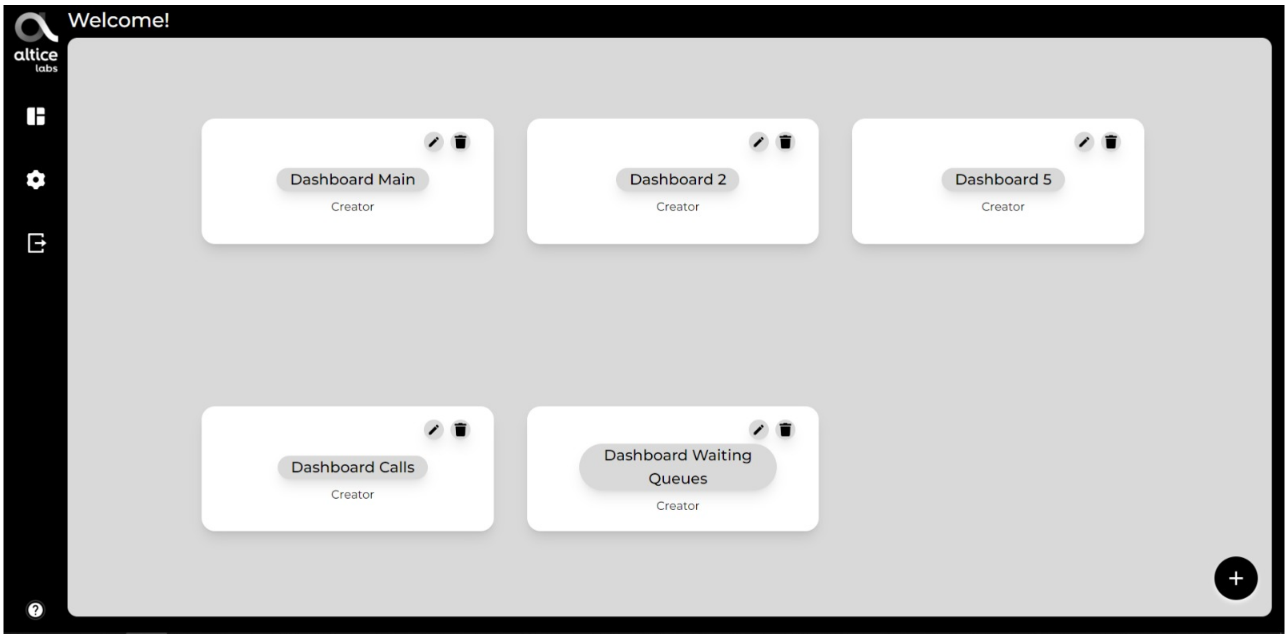


Figure 5.21: Print-screen of the Main Page of the web application

Figure 5.21 showcases a screenshot captured directly from the web application, displaying the main page. The layout consists of a side bar and a black bar positioned at the corners of the screen. The main content of the page is loaded in the center.

The main page utilizes the dashboard cards component, allowing users to interact with their dashboards. These cards provide functionality such as opening, editing, and deleting dashboards. Additionally, the floating action button (FAB) is available for users to create new dashboards, providing a convenient and quick way to initiate the creation process.

The main page serves as a central hub for users to manage and access their dashboards, facilitating a seamless and intuitive experience. By leveraging the dashboard cards component and the FAB, users can efficiently navigate and interact with their personalized dashboards.

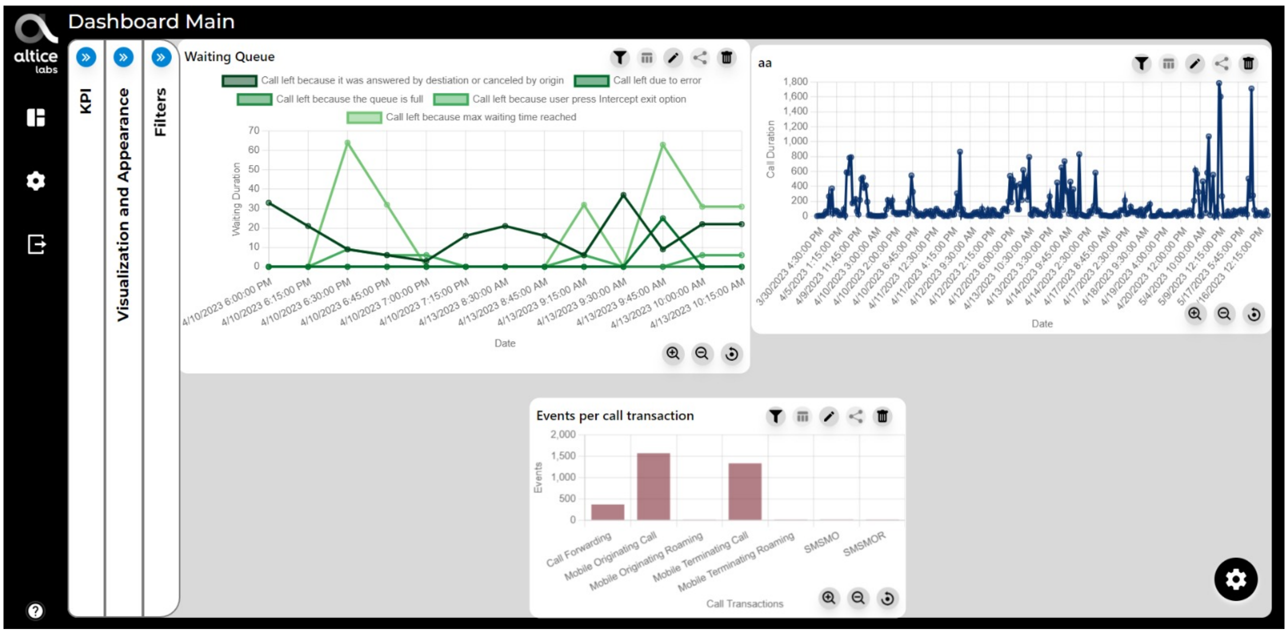
Dashboard Page

The page depicted in Figure 5.22 showcases a dynamic and versatile layout designed to accommodate various components and functionalities. Composed of three toolbar components, as illustrated in Figure 5.23, the page offers a range of tools and options to enhance the user experience.

Each toolbar component incorporates dropdown components, which provide users with access to additional features and settings. The dropdowns consist of draggable components, allowing users to effortlessly select and manipulate elements. The dropzone serves as a designated area for placing axis and filter components, facilitating the creation and customization of widgets.

The filter cards, as mentioned earlier, play a crucial role in enabling users to refine and narrow down data based on specific criteria. The palette picker, situated within one of the toolbars, offers a selection of color ranges, empowering users to customize the visual representation of their widgets.

The central focus of the page lies in the widget cards component. Positioned prominently in the center, these cards serve as the primary elements for displaying data and facilitating user interaction. Users have the freedom to arrange and configure the widget cards according to their preferences and requirements.



Dashboard Main
altice labs
KPI
Visualization and Appearance
Filters
Waiting Queue
Call left because it was answered by destiation or canceled by origin
Call left due to error
Call left because the queue is full
Call left because user press Intercept exit option
Call left because max waiting time reached
Waiting Duration
Date
aa
Call Duration
Date
Events per call transaction
Events
Call Forwarding
Mobile Originating Call
Mobile Originating Roaming
Mobile Terminating Call
Mobile Terminating Roaming
SMSMO
SMSMOR
Call Transactions

Figure 5.22: Print-screen of the Dashboard Page of the web app

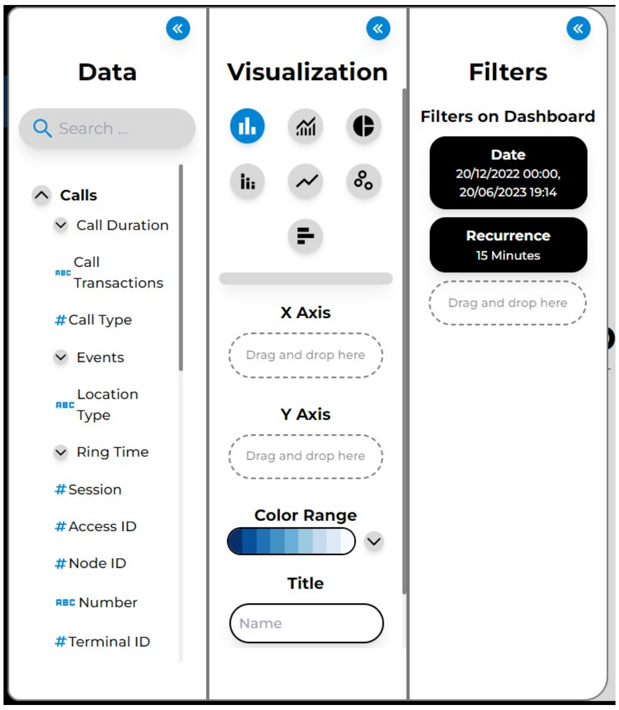
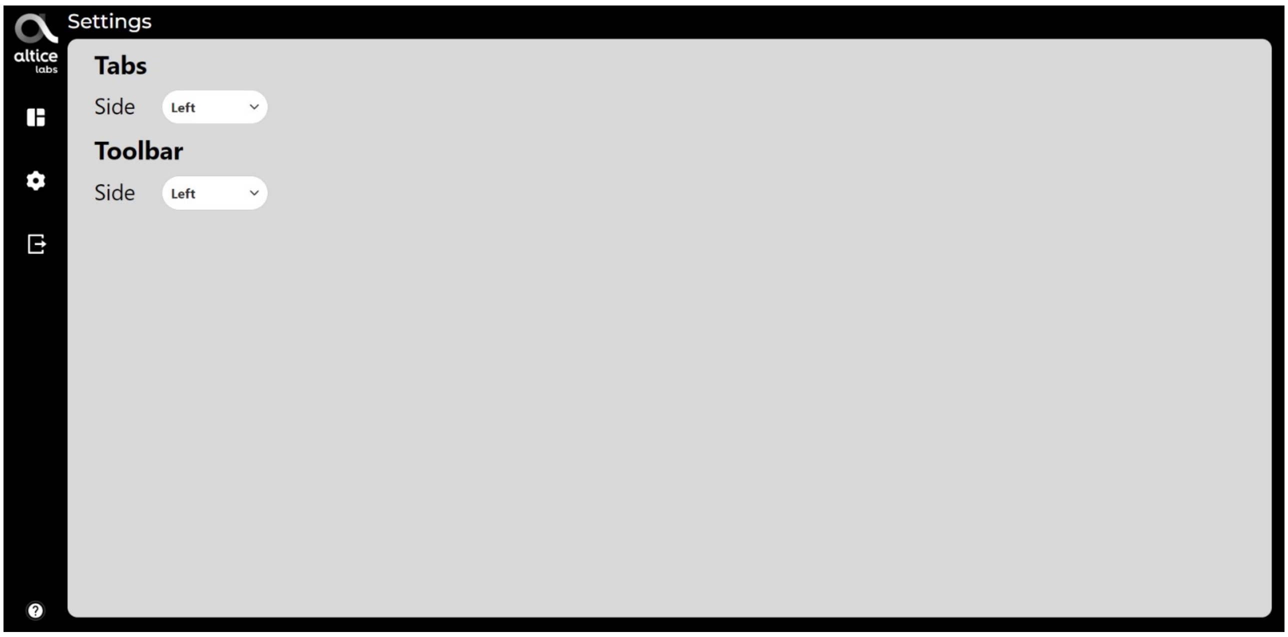


Figure 5.23: Print-screen of the three toolbars open

Settings Page

Figure 5.24 showcases the settings page, which provides users with the ability to customize the layout of the web application. The page consists of two selectors, as depicted in the figure, allowing users to make personalized choices and adjustments.


Settings

Tabs

Side: Left

Toolbar

Side: Left

Figure 5.24: Print-screen of the Settings Page

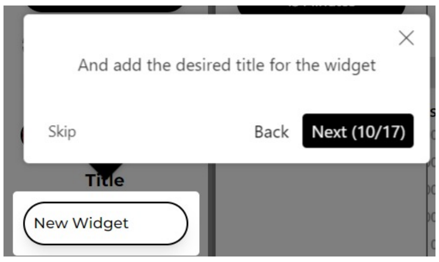
Tutorial

To facilitate the learning process for users, a tutorial feature was implemented within the web application. This tutorial utilizes the React Joyride package, which allows for interactive walkthroughs to guide users through the various steps and functionalities of the application.

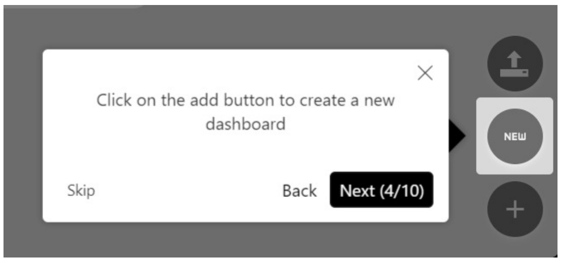
Figure 5.25 showcases three examples of the walkthrough process. The tutorial systematically guides users through each step involved in creating a widget, starting with the creation and management of a dashboard. Users are provided with clear instructions on how to navigate the dashboard page and perform actions such as adding metrics to the axis and applying filters.

The tutorial also emphasizes the interactive nature of the application by demonstrating how users can interact with the charts within the widget. It highlights the ability to resize and move widget cards to customize their placement on the page.

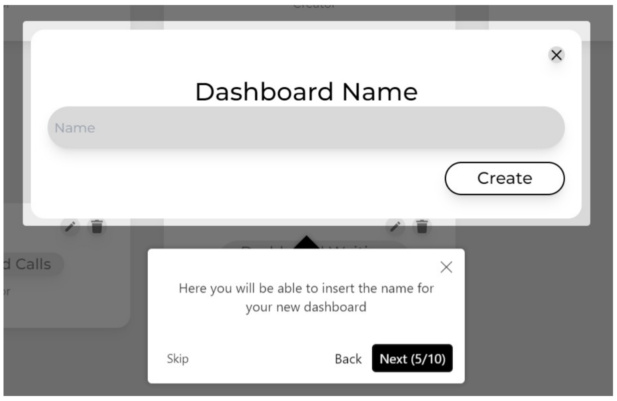
By incorporating the tutorial feature, the web application aims to ensure that users can quickly familiarize themselves with the functionalities and workflow of the platform. The interactive walkthroughs provide step-by-step guidance, empowering users to efficiently navigate and utilize the various features offered by the application.


And add the desired title for the widget

(a)


Click on the add button to create a new dashboard

(b)



Here you will be able to insert the name for your new dashboard

(c)

Figure 5.25: Screen-shots of parts of the walktrough tutorial

Chapter 6

Tests

Testing is an essential aspect of software development to ensure the quality, functionality, and usability of the application. In the case of this web application, various types of tests were performed, including unit tests and usability tests.

Unit tests were conducted to verify the correctness and functionality of individual units or components of the application. These tests focused on testing isolated endpoints to ensure that they behaved as expected. Unit tests helped to identify and fix any bugs or errors in the code early in the development process, promoting code reliability and maintainability.

Usability tests were also conducted to evaluate the user experience and interface design of the web application. These tests involved real users performing tasks and providing feedback on the usability, intuitiveness, and effectiveness of the application. Usability tests helped to identify any usability issues, navigation problems, or areas of improvement in the user interface.

6.1 API Testing

To validate the functionality and accuracy of the API endpoints, a black box testing method was employed. This approach involved testing the API without detailed knowledge of its internal workings, focusing solely on the inputs and outputs.

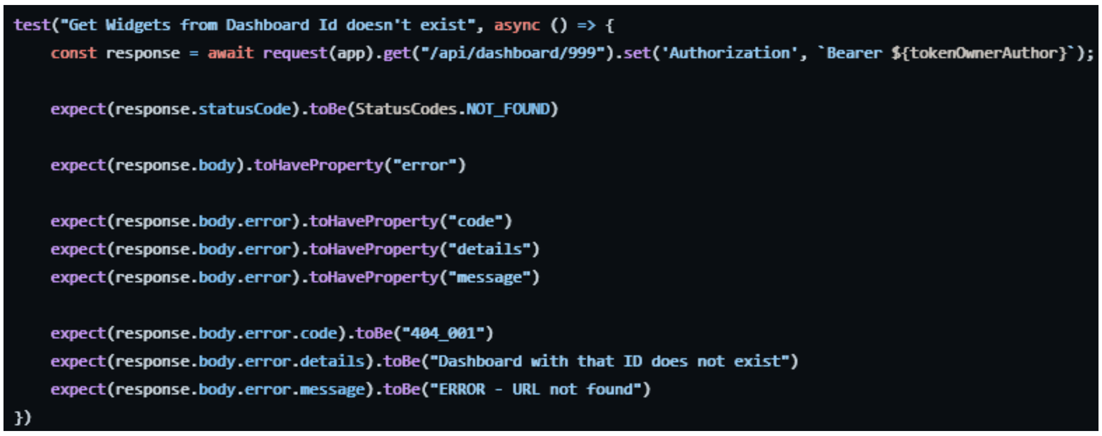
For each endpoint, the output generated by the API was compared against the expected outcome. This comparison served as a means to verify that the API was functioning correctly and producing the desired results.

The API development followed an API-first approach, meaning that the anticipated output format was predetermined during the creation of the endpoints.

This streamlined the testing process, allowing for a comparison between the expected and actual outputs.

To automate the testing procedure, Jest was utilized as the testing framework. Jest is a popular JavaScript testing framework that offers a straightforward and effective way to create tests.

Figure 6.1 illustrates an example of a test case. The test case involves sending a request to a specific API endpoint and comparing the response with the expected result. The response can be examined for the correct status code, data format, and content. The Jest framework provides various assertion methods to facilitate these comparisons.



```
test("Get Widgets from Dashboard Id doesn't exist", async () => {
    const response = await request(app).get("/api/dashboard/999").set('Authorization', `Bearer ${tokenOwnerAuthor}`);

    expect(response.statusCode).toBe(StatusCodes.NOT_FOUND)

    expect(response.body).toHaveProperty("error")

    expect(response.body.error).toHaveProperty("code")
    expect(response.body.error).toHaveProperty("details")
    expect(response.body.error).toHaveProperty("message")

    expect(response.body.error.code).toBe("404_001")
    expect(response.body.error.details).toBe("Dashboard with that ID does not exist")
    expect(response.body.error.message).toBe("ERROR - URL not found")
})
```

Figure 6.1: Example of a Jest test code

To simplify testing, a separate database was employed, and a bash script was generated to populate the database with test data prior to executing the test procedures. Once the tests were completed, the script would clean up and remove all the test data from the database, ensuring a pristine and consistent testing environment for each test run. The structure was as followed:

- **Initial setup**: A bash script was responsible for creating the necessary resources to facilitate the tests;
- **Pass parameters to request**: The tests were designed to utilize the resources created in the initial setup. The built-in fetch method was used to make the requests, and a specific token, which does not expire, was generated solely for testing purposes;
- **Asserting body of response**: The response body was compared to the expected outcome using Jest's methods, such as "toHave";
- **Asserting data**: In addition to the response body, the data itself needed to be validated to ensure the correct processing. Jest's "toBe" method was employed for this purpose.

Strategy

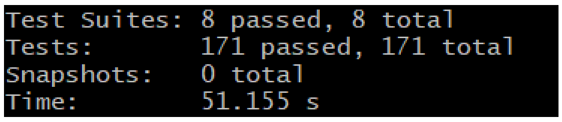
To ensure consistency and reliability in the testing process, a testing strategy was followed. This strategy included the following key aspects:

- **Test authentication**: Tests were performed to verify if it was possible to access the endpoints without proper authentication. Requests without a valid token or with an invalid token were made to ensure that unauthorized access was properly restricted;
- **Test authorization**: Tests were conducted to validate if users with insufficient permissions were prevented from accessing or modifying the resources. This ensured that the access control mechanisms were correctly implemented;
- **Test expected behavior**: The tests focused on verifying that the API endpoints produced the expected results. This involved comparing the actual output with the expected output, such as checking the correctness of returned data, response status codes, and data formats;
- **Test invalid requests**: Tests were designed to ensure that the API responded correctly to invalid requests. This involved checking if the API returned the appropriate error codes and error messages for various scenarios, such as missing parameters or invalid input.

To streamline the development of tests, the test files were organized into eight test suites, with each suite corresponding to a specific group of endpoints. This division aided in the management and organization of the testing process.

Results

A comprehensive total of 117 tests were executed on all endpoints, utilizing the described strategy. Figure 6.2 provides an overview of these tests, demonstrating that all test suites passed successfully. This outcome indicates that the tests verified the expected response bodies and data for each endpoint. However, it is important to note that passing tests does not guarantee a bug-free program; it solely affirms that the program is functioning as intended according to the provided test cases. The tests conducted, along with their corresponding labels, are documented in Appendix D.



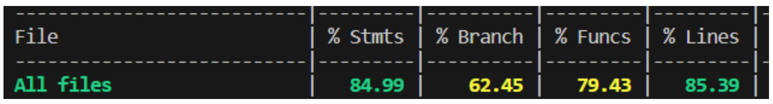
```
Test Suites: 8 passed, 8 total
Tests:       171 passed, 171 total
Snapshots:   0 total
Time:        51.155 s
```

Figure 6.2: Tests done

Figure 6.3 provides an overview of the code coverage for all the API tests. Code coverage is a white-box technique that measures the extent to which the source code of a program is exercised by the tests. It helps identify areas of the code that have not been adequately tested, including functions, branches, and lines.

In the image, the "%Smts" represents the percentage of covered statements during the test execution. "%Branch" indicates the percentage of tested branches, which measures the decision-making statements followed by the tests. "%Funcs" represents the percentage of functions tested, while "%Lines" indicates the percentage of covered lines in the code.

Based on the obtained results, it can be observed that the percentage of covered lines and statements exceeds the expected threshold of 80%. The percentage of tested functions is also approaching the desired level. However, the coverage for branches is lower than expected, indicating that more thorough testing of decision-making statements is needed in the future of the project.



```
----------------------------|---------|----------|---------|---------|-
File                        | % Stmts | % Branch | % Funcs | % Lines |
----------------------------|---------|----------|---------|---------|-
All files                   |   84.99 |    62.45 |   79.43 |   85.39 |
```

Figure 6.3: Code Coverage

6.2 Frontend Testing

In this section, we will present and explain the testing conducted on the frontend as well as the usability tests.

6.2.1 Functionality Testing

To adhere to AlticeLabs QA team guidelines, only manual tests were conducted for this version of the project. The decision to prioritize manual testing was made to ensure that the most essential features were thoroughly tested before moving on to automated testing. It was agreed that for future versions, the QA team would begin writing and implementing automated tests.

The manual tests were define in a csv in order to import it to XRay. Xray is a test management application that integrates with Jira and SVN and allows to easly create issues in case of bugs occuring. This way tests are ready to follow for the next stage, that will be conducted by the QA team.

These initial tests focused on identifying any issues related to the integration of each page's functionalities with their respective endpoints. The primary objective was to ensure that the functionalities implemented in the frontend pages were effectively integrated with the designated API endpoints. This allowed for early identification and resolution of any issues that could hinder the proper functioning of the system as a whole.

Authentication

Four tests were defined to ensure the authentication process functions correctly in the web application. The tests are described below, in table 6.2, along with their expected results:

Test Case #	Name	Steps	Expected Result	Result
1	Login Page	1. Clear cache 2. Open Web App	Login page should appear	Pass
2	Login	1. Open Web App 2. Insert valid login details	Redirect to main page	Pass
3	Invalid Login	1. Open Web App 2. Insert invalid login details	Stay on login page and warn user about incorrect login	Pass
4	Logout	1. Be logged in 2. Click on the logout button	Redirect to login page	Pass

Table 6.1: Authentication Manual Tests

These tests ensure that users cannot access the website without proper authorization, and when entering the application for the first time or after logging out, they are redirected to the login page. Additionally, the platform correctly warns the user when incorrect login details are provided.

Settings

The settings page was tested to verify if the changes made by users resulted in the expected UI modifications. Specifically, the side bar and the tabs of the dashboard were tested to ensure proper functionality and visual updates based on user settings:

Test Case #	Name	Steps	Expected Result	Result
1	Change side bar to right side	1. Open Web App 2. Click on Settings button on the navbar 3. Select "Right" on the dropdown selection for the navbar	Navbar should change to the right side along with all the complementary buttons changing to the left	Pass

2	Change side bar to left side	1. Open Web App 2. Click on Settings button on the navbar 3. Select "Left" on the dropdown selection for the navbar	Navbar should change to the left side along with all the complementary buttons changing to the right	Pass
3	Change dashboard tabs to the right side	1. Open Web App 2. Click on Settings button on the navbar 3. Select "Right" on the dropdown selection for the tabs	Dashboard tabs should change to the right side along with all the complementary buttons changing to the left	Pass
4	Change dashboard tabs to the left side	1. Open Web App 2. Click on Settings button on the navbar 3. Select "Left" on the dropdown selection for the tabs	Dashboard tabs should change to the left side along with all the complementary buttons changing to the rle	Pass

Table 6.2: Settings Manual Tests

Dashboard

A series of tests were conducted to ensure the proper functionality of the dashboard-related features. The detailed test cases can be found in Table E.1 in Appendix E. This section provides an overview of the testing performed, including additional context and information. All of the tests done were passed.

The tests encompassed various aspects of dashboard management and functionality, such as creating a new dashboard, editing existing dashboards, and deleting dashboards. These tests focused on validating the system's response to different scenarios, such as empty dashboard names during creation and editing processes.

Furthermore, the tests examined the behavior of dashboards when accessed by different user roles, including authors, readers, and editors. The goal was to ensure that the appropriate permissions and functionality were granted to users based on their assigned roles.

The sharing functionality of dashboards was also thoroughly tested. This involved sharing dashboards with other users in different roles, such as sharing in "Editor" mode, where the shared user could access and edit the dashboard, and sharing in "Viewer" mode, where the shared user could only access and apply filters to the dashboard without editing the widgets.

Additional tests were conducted to assess the effectiveness of applying filters and modifying dashboard settings. These tests examined the ability to change the start and end dates for widgets, modify recurrence data to display data on different time intervals, and add global filters to restrict the data displayed by the widgets.

Overall, the tests aimed to ensure the proper functionality, usability, and security of the dashboard features, providing a robust and reliable experience for users.

Widget

A series of tests were conducted to ensure the proper functionality of widget-related features. The detailed test cases can be found in Table E.2 in Appendix E. This section provides an overview of the testing performed, including additional context and information.

The tests focused on various aspects of widget management and functionality, such as creating different types of widgets, editing widget properties, and performing actions like moving, resizing, and filtering widgets.

The creation of each widget type was thoroughly tested following the standard creation process. It was verified whether the widgets were successfully created and displayed correctly on the dashboard. Additionally, the system's ability to handle invalid inputs during widget creation was also tested, ensuring that appropriate error messages were triggered.

Widget editing capabilities were evaluated through tests involving actions like moving and resizing widgets. The expected result was that the widgets would maintain their new positions and sizes even after a refresh, ensuring the persistence of changes.

Furthermore, tests were conducted to validate the functionality of adding filters to widgets. The aim was to ensure that the widgets would display data specific to the selected users, while removing filters should result in the widget showing all available data.

The ability to edit various widget properties, such as the title, color range, and axis, was also thoroughly tested. The expected result was that changes made to these properties would be successfully reflected in the widgets on the dashboard.

To assess the deletion of widgets, tests were performed to verify that widgets could be successfully removed from the dashboard, ensuring proper clean-up and management of widgets.

Tests were also conducted to evaluate the effectiveness of the search functionality, both for data and filters on the dashboard. The expected result was that searching for specific terms would display the relevant fields containing the search term.

Overall, the tests aimed to ensure the proper functionality, usability, and consistency of widget features, providing users with a reliable and intuitive experience when creating, editing, and managing widgets on the dashboard.

6.2.2 Usability Testing

To assess the usability of the system, the System Usability Scale was utilized. The SUS is a subjective measure of usability, consisting of 10 questions with five response options each, following the Likert response format. It was developed by John Brooke [12]. While the SUS does not guarantee usability, it provides a quick and simple way to evaluate it and offers valuable insights.

The ten questions used in the SUS are as follows:

- I think that I would like to use this system frequently;
- I found the system unnecessarily complex;
- I thought the system was easy to use;
- I think that I would need the support of a technical person to be able to use this system;
- I found the various functions in this system were well integrated;
- I thought there was too much inconsistency in this system;
- I would imagine that most people would learn to use this system very quickly;
- I found the system very cumbersome to use;
- I felt very confident using the system;
- I needed to learn a lot of things before I could get going with this system.

The testers were given a short time frame of five seconds to provide their answers, ensuring that the responses were based on intuition and initial impressions of the application.

After collecting the responses, a score ranging from 0 to 100 was calculated. Following the standards set by Bangor et al. [8], a score of 85 indicates exceptional usability, while anything below 70 is considered unacceptable. The scoring methodology is as follows:

- Each item response is values from 0 to 4;
- Odd numbered questions values is the scale position minus 1;
- Even numbered questions is 5 minus the scale position;
- The sum of the scores is multiplied by 2.5.

In addition to the questionnaire, a set of tasks was defined to further evaluate the system. These tasks included:

- 1- Follow the walktrough tutorial;
- 2- Create a new dashboard and open it;
- 3- Create a widget;
- 4- Edit the widget created;
- 5- Filter a widget;
- 6- Edit settings.

The time taken to complete each task as well as the number of errors was recorded to analyze task performance.

The usability test was conducted with 5 participants, all of whom were members of AlticeLabs and had no prior knowledge or information about the project. This approach ensured that the testers had a genuine first-time experience with the platform, providing valuable insights into its usability. One of the advantages of using the SUS is that even with a small number of testers, such as 5 participants, it is possible to obtain meaningful results for an early-phase usability study [57].

Results

The results of the questionnaire provided by each tester are presented in Table 6.3.

User	Total Score
1	82.5
2	90
3	80
4	75
5	87.5
Mean	83

Table 6.3: Results of the SUS

Upon observing the results, it is evident that the average score surpasses the threshold of 70, indicating a high level of usability for the tested system. This outcome suggests that the majority of users perceived the system as user-friendly, intuitive, and easy to navigate. These results are a testament to the successful implementation of a usability-focused approach during the design phase.

While it is important to note that the SUS scores alone may not serve as the sole determinant of overall usability, it serves as a valuable starting point for measuring user satisfaction. By achieving a noteworthy average score, it is clear that the system's design considerations effectively contributed to a positive user experience.

In Table 6.4, the average time taken by users to complete each task and the corresponding percentage of errors committed are presented. An error is defined as an unintentional action performed by the user while attempting to carry out the correct task, such as clicking the wrong button or dragging an item to the wrong location. The percentage was calculated by the number of wrong actions divided by the total of actions multiplied by 100.

Task	Mean Time (Minutes:Seconds)	Mean Percentage of Errors
1	03:25	0%
2	00:26	7%
3	01:27	13%
4	00:23	8%
5	00:38	9%
6	00:30	0%

Table 6.4: Means of the times and number of errors for each task

The complete tutorial had an average duration of 3 minutes and 25 seconds, which is below the specified requirement of 5 minutes outlined in the non-functional requirements. Additionally, following the tutorial, all users successfully completed the assigned tasks, resulting in a 100% success rate.

Furthermore, each task exhibited a mean execution time of under two minute, aligning with the other defined non-functional requirement.

As anticipated, the creation and filtration of widgets emerged as the most time-consuming tasks. This outcome was expected due to the inherent complexity associated with these core functionalities of the platform. However, it is noteworthy that all testers were able to create various types of widgets with different attributes, such as colors, titles, types, and charts. This indicates that even users with limited experience can easily navigate and utilize the application after undergoing the tutorial.

The UI customization process required an average of three clicks. This signifies that the customization feature is as user-friendly and easily identifiable as intended, aligning with the planned design considerations.

Although the number of users involved in the testing process was limited, it is possible to conclude that the application has demonstrated promising usability characteristics, as it adheres to usability-focused development principles. However, further user testing with a larger and more diverse group of participants will provide more comprehensive insights to further refine and enhance the application's usability.

Chapter 7

Conclusion

7.1 Final Status

To determine the overall success of the project, it is crucial to refer to the defined threshold of success outlined in Chapter 2.

Upon evaluation, it can be concluded that the project has met the threshold of success for the following reasons:

- The Must Have functionalities, as well as the Should Have functionalities, were successfully developed, reviewed, documented, tested, and deployed. This accomplishment fulfills the first point of the threshold;
- As displayed in section 6 the users had over the 85% accuracy rate on the tasks given to them and completed them all under/over a minute each. The tutorial took on average X minutes and had a 100% success rate. The customziation had the expected 3 clicks. This means that the usability non-functional requirements all passed;
- The non-functional requirements for maintainability were fully satisfied. The frontend demonstrates a reusability rate of 84.6%, surpassing the defined threshold of 80%. Additionally, the line coverage rate stands at 85.30%, exceeding the minimum requirement of 80%, the documentation of the frontend components can be found in appendices C, an excerpt of the API documentation can be found in figure 5.2;
- All the restrictions were followed.

With the evidence presented, it is possible to confidently conclude that the project was indeed a resounding success. Not only were all the objectives and milestones successfully achieved, but the results surpassed expectations.

7.2 Problems

This section describes the problems faced that slowed down the development of the project.

Hardware and Software

The development process at AlticeLabs was significantly impeded by the outdated and slow computer provided for internal use. The slow performance of the computer, including a slow Integrated Development Environment (IDE) and lengthy compiling times, hindering the productivity.

Moreover, accessing the AlticeLabs network required the use of a VPN (Virtual Private Network). Unfortunately, this VPN encountered connectivity issues, often disconnecting unexpectedly as a result, time was wasted daily while attempting to establish and maintain a connection to the network. Furthermore, it was incompatible with the eduroam network.

Communication

Communication within the team members proved to be efficient and prompt, facilitating smooth collaboration. However, when it came to communication with members from other teams, delays were experienced, resulting in significant challenges.

Specifically, meetings with other teams were required for tasks such as database redesign or confirming KPIs and mockups. Unfortunately, these meetings encountered delays, which subsequently impacted the development process.

Deployment

In addition to the deployment issues mentioned earlier, such as the unprepared deployment machine for the applications, there was a lack of an automated pipeline. This meant that after the initial deployment was ready, subsequent deployments required manual creation of build files and their transfer to the machine via FTP.

7.3 Future Work

This section focuses on the future work that can be done with the project as it will continue to be develop by other AlticeLabs team members.

7.3.1 Additional Features

The application developed represents only an MVP (Minimum Viable Product) of the final product, implying that there is ample room for further improvements and feature development.

Could Have Requirements

The upcoming implementations can focus on incorporating the "Could Have" features outlined in the defined requirements.

One important step would be implementing a pagination system for the dashboard. The groundwork for this feature has already been laid, with both the database and API being prepared to handle the necessary information.

One significant enhancement would be the addition of specific charts based on user-selected metrics, as this would greatly enhance usability. Many similar platforms have already implemented this feature into their systems. Additionally, enabling email reports for the dashboard would allow users to set thresholds and receive alerts when these thresholds are surpassed. This feature would greatly benefit users and their monitoring needs.

One major feature to consider is the ability to drag and drop metrics onto the axes, ensuring that they do not cause any conflicts. This involves leveraging the existing database division of metrics and dimensions to determine the suitable placement of each metric and dimension on the chart's axes.

Furthermore, introducing drilldown functionality would provide users with a deeper level of interaction with the system, thereby enhancing the user experience and ease of use.

Annotations

The ability for dashboard users to create annotations serves as a valuable feature that facilitates idea generation and collaboration. However, there is potential for further enhancements to maximize the value and utility of annotations within the dashboard.

Future implementations can focus on expanding the functionality of annotations to foster increased information sharing and collaboration among users. This could include features such as the ability to mention and notify specific users or teams within annotations, allowing them to receive notifications and actively participate in the discussions or provide further insights.

Additional Chart Types

The inclusion of additional types of charts in the dashboard will offer future users a wider range of options for creating their widgets. This expansion of chart types contributes to improved usability and opens up new possibilities for various use cases.

By introducing diverse chart types, users will have the flexibility to visualize data in different ways, catering to their specific needs and preferences.

These additional chart options not only enhance the visual appeal of the dashboard but also offer more comprehensive and nuanced data representations. Users can select the most suitable chart type based on the nature of their data and the insights they wish to convey.

Artificial Intelligence

Considering the rapidly evolving nature of the dashboard market, new features will continuously become available. One potential future addition could be the implementation of Artificial Intelligence (AI) capabilities. This would enable the AI to generate widgets based solely on user-provided descriptions, adding a level of automation and customization to the application.

7.3.2 Tests

There is significant room for improvement in the testing phase. Firstly, the frontend would benefit from the implementation of automated tests to provide more thorough bug confirmation. Additionally, the API should undergo a more rigorous testing process, extending beyond black box unit tests. Conducting load and performance testing in the future is essential to ensure scalability for increased user numbers.

While the SUS method provides a good indicator of usability, it alone is not sufficient. Further usability tests should be conducted with a larger pool of users, including potential buyers of the application. This is the most effective way to ensure that the application truly delivers a high level of usability. By incorporating feedback from diverse users, it becomes possible to identify areas for improvement and make the necessary refinements to enhance the overall user experience.

Bug Fixing

The application requires further bug fixing to ensure its smooth functionality. A comprehensive list of known bugs has already been shared with the development team to provide clarity on the necessary changes. The identified issues encompass several areas that need attention.

Optimization

Once performance testing on the API is completed, it is recommended to focus on optimizing the less efficient modules of the code. This optimization can be achieved by enhancing the performance of database queries and optimizing functions that access and manipulate data.

In addition to code optimizations, implementing a better caching system within the database can bring further performance improvements. By implementing an effective caching mechanism, the application can retrieve data more quickly, resulting in faster response times for accessing, editing, or creating new charts.

7.4 Final Thoughts

I consider the internship to be a resounding success and a highly positive experience. All the expected functionalities were successfully implemented, surpassing initial expectations and leaving the system in an excellent state, despite the future work that still lies ahead.

From a technical perspective, I have acquired a wealth of knowledge throughout the internship. Even though it was a solo project, being part of a real-world enterprise team has provided invaluable insights into how teams collaborate and communicate effectively. I have deepened my understanding of developing APIs, learning the importance of designing and architecting an API properly. Moreover, I have fulfilled my long-standing aspiration to work with React, expanding my skill set in front-end development. Additionally, I took my first steps into the realm of automated testing, gaining valuable experience in this critical aspect of software development.

The feedback received from both team members and non-team members has been instrumental in my professional growth. This constructive feedback has not only honed my technical skills but has also contributed to my personal development as an individual.

Overall, this internship has been a remarkable journey of learning, growth, and accomplishment. It has provided me with invaluable practical experience in a professional setting, fostering a deeper understanding of teamwork, technical implementation, and the iterative nature of software development. I am immensely grateful for the opportunity and look forward to applying the knowledge and skills acquired during this internship to future endeavors.

References

[1] The investment in ucaas will reach 96,000 million dollars in 2023?, 2017. URL https://enreach.es/en/blog/the-investment-in-ucaas-will-reach-96000-million-dollars-in-2023/. [Last accessed 23/11/2022].

[2] About 8x8 contact center dashboards, 2022. URL https://docs.8x8.com/8x8WebHelp/8x8Analytics/Content/8x8Analytics/Dashboards.htm. [Last accessed 23/11/2022].

[3] 8x8. Create a dashboard using widgets. URL https://docs.8x8.com/8x8WebHelp/8x8Analytics/Content/8x8Analytics/Create_Dashboards.htm. [Last accessed 08/06/2023].

[4] ISO 9241-11:2018. *Ergonomics of human-system interaction — Part 11: Usability: Definitions and concepts*. 2018.

[5] S. Taing A. Picot, K. Riemer. Unified communications. 2008.

[6] Albright, S.C., Winston, W., and Zappe. *Data Analysis and Decision Making*. Cengage Learning, 2010.

[7] B. Bach, Z. Wang, M. Farinella, D. Murray-Rust, and N. Henry Riche. Design patterns for data comics. pages 1–12, 2018.

[8] Kortum P. Miller J.A Bangor, A. The system usability scale (sus): An empirical evaluation. 2008.

[9] Alfie Abdul-Rahman Cagatay Turkay Saiful Khan Yulei Fan Benjamin Bach, Euan Freeman and Min Chen. Dashboard design patterns. 2022.

[10] Richard Brath and Michael Peters. Dashboard design: Why design is important. 2004.

[11] D. Brodbeck and L. Girardin. Design study: Using multiple coordinated views to analyze geo-referenced highdimensional datasets. pages 104–111, 2003.

[12] John Brooke. Sus: A quick and dirty usability scale. 1995.

[13] Simon Brown. *The C4 model for visualising software architecture*. 2022.

[14] S. Faulkner C. Burnay, S. Bouraga and I. Jureta. User-experience in business intelligence - a quality construct and model to design supportive bi dashboards. 2020.

[15] J. Van Biljon C. Jooste and J. Mentz. Usability evaluation for business intelligence applications: A user support perspective. 2014.

[16] Bernardita Calzon. Make sure you know the difference between strategic, analytical, operational and tactical dashboards, 2021. URL `https://www.datapine.com/blog/strategic-operational-analytical-tactical-dashboards`. [Last accessed 06/02/2023].

[17] S. Card and J. Mackinlay. The structure of the information visualization design space. pages 92–99, 1997.

[18] H. Chen. Toward design patterns for dynamic analytical data visualization. 5295:75–86, 2004.

[19] Kolade Chris. Rest api best practices – rest endpoint design examples. URL `https://www.freecodecamp.org/news/rest-api-best-practices-rest-endpoint-design-examples/`. [Last accessed 17/06/2023].

[20] Alistair Cockburn. *Writing Effective Use Cases*. 2000.

[21] Datapine. How to use dashboard filters | datapine. URL `https://www.youtube.com/watch?v=z8relzC_Tqc`. [Last accessed 08/06/2023].

[22] DialPad. Call analytics. URL `https://www.dialpad.com/features/call-analytics/`. [Last accessed 08/06/2023].

[23] A. Djurovic. What uc is and isn't, 2021. URL `https://www.techtarget.com/searchunifiedcommunications/definition/unified-communications`. [Last accessed 16/10/2022].

[24] W. W. Eckerson. *Performance dashboards: measuring, monitoring, and managing your business*. John Wiley & Sons, 2010.

[25] Ferdio. Data viz project. URL `https://datavizproject.com/`. [Last accessed 08/06/2023].

[26] S. Few. *Information Dashboard Design: The Effective Visual Communication of Data*. 2006.

[27] Stephen Few. Dashboard confusion. 2004.

[28] Stephen Few. Practical rules for using color in charts. 2008.

[29] Maria-Elena Froese and Melanie Tory. Lessons learned from designing visualization dashboards. 36(2):83–89, 2016.

[30] Sherya Gate. Using session cookies vs. jwt for authentication. URL `https://hackernoon.com/using-session-cookies-vs-jwt-for-authentication-sd2v3vci`. [Last accessed 17/06/2023].

[31] B. G. Glaser and A. L. Strauss. The discovery of grounded theory: Strategies for qualitative research. 2017.

[32] M. Heitzler H.-J. Schulz, T. Nocke and H. Schumann. A design space of visualization tasks. 19(12):2366–2375, 2013.

[33] S. He and E. Adar. Vizitcards: A card-based toolkit for infovis design education. 1:561–570, 2017.

[34] Bakker M.A. Li S. Kraska T. Hidalgo C. Hu, K. Vizml: a machine learning approach to visualization recommendation. page 128.

[35] J. Hullman and B. Bach. Picturing science: Design patterns in graphical abstracts. pages 183–200, 2018.

[36] J. Kohlhammer J. Bernard, D. Sessler and R. A. Ruddle. Using dashboard networks to visualize multiple patient histories: a design study on post-operative prostate cancer. 25(3):1615–1628, 2018.

[37] Michael Bostock Jeffrey Heer and Vadim Ogievetsky. A tour through the visualization zoo. 2010.

[38] P. Kaur and M. Owonibi. A review on visualization recommendation strategies. 2017.

[39] Howe B. Perry D. Aragon C. Key, A. Vizdeck: self-organizing dashboards for visual analytics. pages 681–684.

[40] Yuhao Li. The case analysis of the scandal of enron. 5(10), 2010.

[41] Looker. Using sql runner to create derived tables. URL `https://cloud.google.com/looker/docs/sql-runner-create-derived-tables`. [Last accessed 08/06/2023].

[42] Hanrahan P. Stolte C. Mackinlay, J. Show me: automatic presentation for visual analysis. 6(13):1137–1144, 2007.

[43] S. M. Magnus and A. Rudra. *Operationally Intuitive Logistics Dashboards for Supply Chain Management in Oil and Gas Based on Human Cognition*, volume 20. Nov. 2019.

[44] Geshan Manandhar. Organizing your express.js project structure for better productivity, 2022. URL `https://blog.logrocket.com/organizing-express-js-project-structure-better-productivity/`. [Last accessed 17/06/2023].

[45] Gina Narcisi. Gartner ucaas magic quadrant: The top 12 vendors in 2019 as at&t, verizon and others are dropped from report, 2019. URL https://www.crn.com/slide-shows/networking/gartner-ucaas-magic-quadrant-the-top-12-vendors-in-2019-as-at-t-verizon-and-othe 6. [Last accessed 23/11/2022].

[46] J. Nielsen. Usability engineering. 1994.

[47] David P. Norton and Robert Kaplan. *The Balanced Scorecard: Translating Strategy into Action*. 1997.

[48] University of New Brunswick. Understanding quality attributes. URL https://www.cs.unb.ca/~wdu/cs6075w10/sa2.htm. [Last accessed 11/06/2023].

[49] M. Plumlee and C. Ware. Integrating multiple 3d views through frame-of-reference interaction. pages 34–43, 2003.

[50] A. A. Rahman, Y. B. Adamu, and P. Harun. Review on dashboard application from managerial perspective. pages 1–5, 2017.

[51] Nihar Raval. React vs angular: Which js framework to pick for front-end development?, 2019. URL https://radixweb.com/blog/react-vs-angular#advantages. [Last accessed 17/11/2022].

[52] Eric Reiss. *Usable Usabilty Simple Steps for Making Stuff Better*. 2012.

[53] Severino Ribecca. The data visualisation catalogue. URL https://datavizcatalogue.com. [Last accessed 08/06/2023].

[54] J. C. Roberts. State of the art: Coordinated & multiple views in exploratory visualization. pages 61–71, 2007.

[55] Eugenie Rumiantseva. Analytical dashboard. URL https://dribbble.com/shots/17580449-Analytical-dashboard. [Last accessed 13/06/2023].

[56] A. Sarikaya, M. Correll, L. Bartram, M. Tory, and D. Fisher. What do we talk about when we talk about dashboards? 25(1), 2019.

[57] Jeff Sauro. 10 things to know about the system usability scale (sus), 2013. URL https://measuringu.com/10-things-sus/. [Last accessed 22/06/2023].

[58] B. Schneiderman. Dynamic queries for visual information seeking. 11:70–77, 1994.

[59] P. Schubert and J. H. Glitsch. Use cases and collaboration scenarios: How employees use socially-enabled enterprise collaboration systems (ecs). 4(2), 2016.

[60] K. Sedig and P. Parsons. Design of visualizations for human-information interaction: A pattern-based framework. 4(1):1–185.

[61] Maureen Stone. Choosing colors for data visualization. 2006.

[62] Tableau. Build a basic view to explore your data. URL https://help.tableau.com/current/pro/desktop/pt-br/getstarted_buildmanual_ex1basic.htm. [Last accessed 08/06/2023].

[63] The Financial Times. Financial times visual vocabulary. URL https://ft-interactive.github.io/visual-vocabulary/. [Last accessed 08/06/2023].

[64] Edward Tufte. Envisioning information. 1990.

[65] Edward R. Tufte. *The Visual Display of Quantitive Information*. Graphics Press, 1983.

[66] RingCentral UK. A guide to user analytics | ringcentral office demo. URL https://www.youtube.com/watch?v=1TCBRoiu0Sg. [Last accessed 08/06/2023].

[67] C. Demiralp V. Dibia. Data2vis: automatic generation of data visualizations using sequence to sequence recurrent neural networks. (39):33–46, 2019.

[68] Keshav Vasudevan. The importance of standardized api design. URL https://swagger.io/blog/api-design/the-importance-of-standardized-api-design/. [Last accessed 17/06/2023].

[69] Wattenberg M. Van Ham F. Kriss J. McKeon M. Viegas, F.B. Manyeyes: a site for visualization at internet scale. 6(13):1121–1128, 2007.

[70] M. Voigt and J. Polowinski. Towards a unifying visualization ontology. 2011.

[71] Pietschmann S. Grammel L. Meißner K. Voigt, M. Context-aware recommendation of visualization components. pages 101–109, 2012.

[72] Colin Ware. *Information Visualization: Perception for Design*. Morgan Kaufmann, 1999.

[73] S. Wexler, J. Shaffer, and A. Cotgreave. The big book of dashboards: Visualizing your data using real-world business scenarios. 2017.

[74] Moritz D. Anand A. Mackinlay J. Howe B. Heer J. Wongsuphasawat, K. Voyager: exploratory analysis via faceted browsing of visualization recommendations. 1(22):649–658, 2015.

[75] O. M. Yigitbasioglu and O. Velcu. A review of dashboards in performance management: Implications for design and research. 13(1):41–59, 2012.

This page is intentionally left blank.

Appendices

Appendix A

Use Case Specification

Use Case Specification for the Dynamic Dashboard:

FR01: Create Dashboard

FR02: Edit Dashboard

FR03: Delete Dashboard

FR04: Create Widget

FR05: Edit Widget

FR06: Delete Widget

FR07: Filtering

FR08: Global Filters

FR09: Share Dashboard

FR10: Specific charts for widgets

FR11: Default mode

FR12: Table Widget

FR13: Interactive filtering

FR14: Drilldowns

FR15: Moving Widget

FR16: Resizing Widget

FR17: Importing Widget

FR18: Exporting Widget

FR19: Importing Dashboard

FR20: Exporting Dashboard

FR21: Email Reports

FR22: Tutorial mode

Table A.1: **FR01: Creating Dashboard**

Use Case 1	Creating Dashboard
Level	Sea
Priority	Must Have
Actor	User
Stakeholder & Interests	Using visual elements like charts, widgets provide an accessible way to see and understand trends, outliers, and patterns in data. It provides an excellent way to present data to non-technical audiences and is way easier and quicker than analyzing the raw data
Trigger	Clicking on "Create new Dashboard"
Pre-Conditions	Logged into platform
Post-Conditions	Dashboard created in user account
Main Scenario	
Actor	**System**
1-Selects 'Create Dashboard' 3-Inserts name for dashboard	2-Presents form to insert name 4-Validates the name 5-Creates and saves dashboard in user profile
Alternative Flow	None
Extension Points	4a. Invalid Name 4a1. Warning message to inform that name is invalid

Table A.2: **FR02: Edit Dashboard**

Use Case 2	Edit Dashboard
Level	Sea
Priority	Must Have
Actor	User

Stakeholder & Interests	The user wants to modify the name of an existing dashboard to provide a more descriptive or up-to-date title. They require a simple method to edit the dashboard name without affecting the dashboard′s content or configuration.
Trigger	User selects the dashboard to be edited
Pre-Conditions	Dashboard created by user exists in the user′s account
Post-Conditions	Dashboard name is successfully updated
Main Scenario	
Actor	**System**
1-User selects the dashboard to be edited	
	2-Displays the current name of the selected dashboard
3-User modifies the dashboard name	
	4-Validates the modified name
5-User saves the changes	
	6-Updates the dashboard name
Alternative Flows	None
Extension Points	4a. Invalid name modification 4a1. Displays an error message indicating the specific issue with the name modification

Table A.3: **FR03: Delete Dashboard**

Use Case 3	Delete Dashboard
Level	Sea
Priority	Must Have
Actor	User
Stakeholder & Interests	The user wants to remove an entire dashboard that is no longer needed or relevant. They require a straightforward method to delete the dashboard without affecting other dashboards or causing unintended consequences. Additionally, they desire confirmation prompts or safeguards to prevent accidental deletion.

Trigger	User selects the dashboard to be deleted
Pre-Conditions	Dashboard exists in the user's account
Post-Conditions	Dashboard is successfully deleted from the user's account
Main Scenario	
Actor	**System**
1-User selects the dashboard to be deleted 3-User confirms the deletion	2-Displays a confirmation prompt to ensure the user's intent to delete 4-Deletes the selected dashboard from the user's account
Alternative Flows	None
Extension Points	None

Table A.4: **FR04: Create Widget**

Use Case 4	Create Widget
Level	Sea
Priority	Must Have
Actor	User
Stakeholder & Interests	The user wants to create accurate and visually appealing charts to effectively analyze and communicate data. They require a user-friendly interface and intuitive controls for creating and customizing the charts. Additionally, they desire flexibility in selecting and configuring axis labels, data series, and other relevant options for each chart type.
Trigger	User selects the option to create a new widget
Pre-Conditions	Have dashboard created and be on the dashboard page
Post-Conditions	Widget created and available for user to see and costumize
Main Scenario	
Actor	**System**
1-Opens middle toolbar	

3-Selects the line chart, bar chart or scatter plot 4-Drags desired KPI metrics onto the axis dropzone 8-Adds title for the widget 9-Selects color range 10-Drags desired filters onto filter dropzone 11-Clicks "Done" button	2-Makes available the dropzone for the metrics and widget filter, chart selector, color range selector and form for the widget title 5-Verifies if the metrics can be used on the dropped axis 12-Verifies data and inserts widget onto the middle of the screen
Alternative Flow	None
Extension Points	5a. Metric cannot be used on axis 5a1. Message warning user that he needs to select other metric 5a. Invalid number of metrics on axis 5a1. Message warning user that he needs to addremove metrics on the axis 12a. Invalid information 12a1. Message warning user with what is wrong

Table A.5: **FR05: Edit Widget**

Use Case 5	Edit Widget
Level	Sea
Priority	Must Have
Actor	User

Stakeholder & Interests	The user wants to modify the existing widgets on the dashboard to reflect updated data or changes in the visual representation. They require an intuitive interface to edit the widget's properties, such as metrics, title, colors, and filters. Additionally, they desire flexibility in modifying the widget without affecting other components on the dashboard.
Trigger	User selects the widget to be edited
Pre-Conditions	Dashboard with existing widgets is available
Post-Conditions	Widget is updated with the user's modifications
Main Scenario	
Actor	**System**
1- User selects the widget to be edited	
	2- Displays the current properties of the selected widget, including metrics, title, colors, and filters
3- User modifies the desired properties of the widget	
	4- Validates the modifications made by the user
5- User saves the changes	
	6- Updates the widget with the user's modifications
Alternative Flows	1a. User clicks on top of the widget to initiate the edit mode 1a1. System activates the edit mode and proceeds to step 2 1b. User clicks on the widget edit button to initiate the edit mode 1b1. System activates the edit mode and proceeds to step 2 1d1. System goes to step 6
Extension Points	4a. Invalid modifications 4a1. Displays an error message indicating the specific issue with the modifications

Table A.6: **FR06: Delete Widget**

Use Case 6	Delete Widget
Level	Sea
Priority	Must Have
Actor	User
Stakeholder & Interests	The user wants to remove unnecessary or outdated widgets from the dashboard. They require a straightforward method to delete widgets without affecting other components or causing unintended consequences. Additionally, they desire confirmation prompts or safeguards to prevent accidental deletion.
Trigger	User selects the widget to be deleted
Pre-Conditions	Dashboard with existing widgets is available
Post-Conditions	Widget is successfully removed from the dashboard
Main Scenario	
Actor	**System**
1-User selects the widget to be deleted	
	2-Displays a confirmation prompt to ensure the user's intent to delete
3-User confirms the deletion	
	4-Removes the widget from the dashboard
Alternative Flow	None
Extension Points	None

Table A.7: **FR07: Filtering**

Use Case 7	Filtering
Level	Sea
Priority	Must Have
Actor	User
Trigger	Clicking on widget filter button
Stakeholders Interests	Users want to make quick analysis of the data they have displayed over a certain time period or between other metrics, filtering enables them to do so without the need to create another widget
Pre-Conditions	Have widget created
Post-Conditions	Widget displaying only data filtered

Main Scenario	
Actor	**System**
1-Clicks 'Filter' button 3-Drags desired metric and drops onto widget filter dropzone 5-Selects desired values	2-Lists metrics available for filtering and shows dropzone for widget filter 4-Obtains all the values available to filter 6-Displays the data on the same widget
Alternative Flows	1a. User clicks on top of the widget to initiate filter mode 1a1. System activates the filter mode and proceeds to step 2
Extension Points	None

Table A.8: **FR08: Global Filters**

Use Case 8	Global Filters
Level	Sea
Priority	Must Have
Actor	User
Trigger	Dropping filter onto global filter zone
Stakeholders Interests	Users want to make quick analysis of the data they have displayed over a certain time period or between departments, global filtering enables them to do so quickly without the need for mor widgets or waste of time
Pre-Conditions	Have widgets created
Post-Conditions	Widgets displaying only data filtered
Main Scenario	
Actor	**System**
1-Opens filter tab 3-Selects desired filters	2-Displays start and end date selector, recurrence selector, other selected filters and dropzone too add more filters 4-Updates data on all widgets
Alternative Flows	None

Extension Points	4a. No data on widget 4a1. Display empty chart

Table A.9: **FR09: Share Dashboard**

Use Case 9	Share Dashboard
Level	Sea
Priority	Should Have
Actor	User
Trigger	User clicks on share button
Stakeholders Interests	Users want to restrict usage of their dashboard so only their designated users can update his created dashboard, while the other users only need to read the dashboards he created and not become overwhelmed with learning another tool
Pre-Conditions	Having dashboard created
Post-Conditions	Users can create or read shared dashboard based on role given
Main Scenario	
Actor	**System**
1-Selects options menu 2-Selects "Share with" 4-Enters user's email 5-Selects role	 3-Show popup window with form to share new user and previosly shared users 6-Verifies user and adds dashboard to user shared
Alternative Flows	4a. Wants to edit previously shared user role 4a1. User selects user to edit role and selects new role, the system will update the role
Extension Points	6a. User doesn't exist 6a1. Warning message indicating that shared user doesnt exist 6b. User already shared 6b1. Warning message indicating that the user has already been shared

Table A.10: **FR10: Specific charts for widgets**

Use Case 10	Specific charts for widgets
Level	Fish
Priority	Could Have
Actor	User
Trigger	Creating a new Widget
Stakeholders Interests	The user doesn't want to lose time setting up a chart in a widget that won't make sense when displayed in the final dashboard
Pre-Conditions	Have the metrics selected
Post-Conditions	Selection of charts available
Main Scenario	
Actor	**System**
1-Selects KPI metrics	2-Checks what data is used in the KPIs 3-Selects charts that the data can be displayed with 4-Presents the user with the charts available
Alternative Flow	3.a. System can't detect what charts can't be used for the selected data 3.a.1. All charts are available for the user to select
Extension Points	None

Table A.11: **FR11: Default mode**

Use Case 11	Default mode
Level	Sea
Priority	Could Have
Actor	User
Trigger	Selecting the *Default Mode* button
Stakeholders Interests	The user has a quick dashboard made while having no work done so he can analyse the KPIs fast if he needs to
Pre-Conditions	Having a blank dashboard
Post-Conditions	Widgets with the data selected
Main Scenario	
Actor	**System**
1-Selects default mode 3-Selects KPIs for dashboard	2-Presents KPIs available

	4-Generates widgets for the selected KPI metrics
Alternative Flows	None
Extension Points	1a. Dashboard not empty 4a1. Message warning user that if he continues all the widgets will be deleted

Table A.12: **FR12: Table Widget**

Use Case 12	Table Widget
Level	Sea
Priority	Could Have
Actor **Trigger**	User Selecting *Table* option in widget
Stakeholders Interests	The user wants to get all the info he has access to, in the form of a table so he can see all the details he wouldn't otherwise
Pre-Conditions	Have a widget created
Post-Conditions	Popup with widget data listed in a table
Main Scenario	
Actor	**System**
1-Selects Widget that he wants to display 2-Selects table button	 3-Selects data from database 4-Presents data in table format
Alternative Flow	None
Extension Points	None

Table A.13: **FR13: Interactive filtering**

Use Case 13	Interactive filtering
Level	Fish
Priority	Should Have
Actor	User
Trigger	Clicking on a value in a widget

Stakeholders Interests	The user has an easier time applying filters and absorbing information when all he is a click away from applying filters to the whole dashboard
Pre-Conditions	Dashboard with widgets
Post-Conditions	Global filter settings update based on the value the user selected
Main Scenario	
Actor	**System**
1-Clicks on widget dataset value	2-Adds value to the global filters 3-Updates data on all widgets
Alternative Flows	None
Extension Points	4a. No data on widget 4a1. Display empty chart

Table A.14: **FR14: Drilldowns**

Use Case 14	Drilldowns
Level	Sea
Priority	Could Have
Actor	User
Trigger	Double-Clicking on a value in a widget
Stakeholders Interests	User has access to the information easier, since he doesn't have to create another widget and can see more information with the use of only one
Pre-Conditions	Having a widget created in the dashboard
Post-Conditions	Widget shows more info about the value selected
Main Scenario	
Actor	**System**
	1- When creating a widget, available drilldowns will be automatically added to the widget
2-Right clicks on specific value from widget	
	3-Displays, on same widget and same chart, more information about the value selected
Alternative Flows	None

Extension Points	1a. Widget doesnt have any drilldown possible 1a1. Informs user that the widget doesnt have any possible drilldown

Table A.15: **FR15: Moving widgets**

Use Case 15	Moving widgets
Level	Sea
Priority	Should Have
Actor	User
Trigger	Dragging Widget
Stakeholders Interests	The user can move the widgets to put the most important ones in the middle where they will get more attention when looking at first glance at the dashboard
Pre-Conditions	Having a widget created
Post-Conditions	Widget on the position defined by the user
Main Scenario	
Actor	**System**
1-Selects widget 2-Drags widget to desired position	3-Saves new position and puts widget on the top layer
Alternative Flows	None
Extension Points	None

Table A.16: **FR16: Resizing widgets**

Use Case 16	Resizing widgets
Level	Sea
Priority	Should Have
Actor **Trigger**	User Selecting and Dragging Widget

Stakeholders Interests	User wants to make the most important widgets bigger in order to highlight it and the less important ones smaller so more widgets can be put on the screen without the need to scroll up and down
Pre-Conditions	Have widget created
Post-Conditions	Widget changes to size defined by the user
Main Scenario	
Actor	**System**
1-Selects corner of widget and resizes to preferred size	2- Saves new widget size and puts widget on the top layer
Alternative Flows	None
Extension Points	None

Table A.17: **FR17: Importing Widget**

Use Case 17	Importing Widget
Level	Sea
Priority	Won't Have
Actor	User
Trigger	Adding Widget
Stakeholders Interests	User wants to add to his dashboard a widget that a coworker made that displays the information he needs without having to rebuild it from the ground up
Pre-Conditions	Have a dashboard created
Post-Conditions	Widget displayed in user dashboard
Main Scenario	
Actor	**System**
1-Creates a new widget 2-Selects import widget 4-Inserts file path	3-Popup window appears where user can insert file path 5-Reads file 6-Loads widget configuration and then loads data from database 7-Inserts widget into dashboard
Alternative Flows	None

Extension Points	5.a Link or file corrupted 5.a.1 Warns user about corruption and doesn't add widget 6.a Can't load data used in the widget from database 6.a.1 Reconnects to database and warns user about wait time bigger than usual

Table A.18: **FR18: Exporting Widget**

Use Case 18	Exporting Widget
Level	Sea
Priority	Won't Have
Actor	User
Trigger	Selecting share option on widget
Stakeholders Interests	User wants to share a widget created by him to his coworker in order to share the information provided by the widget without having to write down how he built the widget
Pre-Conditions	Have a widget created
Post-Conditions	File created with widget configuration
Main Scenario	
Actor	**System**
1-Selects widget to export 2-User selects sharing option	3-Detects what chart, filter and data is used on widget 4-Writes to and downloads file
Alternative Flows	None
Extension Points	None

Table A.19: **FR19: Importing Dashboard**

Use Case 19	Importing Dashboard
Level	Sea
Priority	Won't Have
Actor	User
Trigger	Creating a new dashboard

Stakeholders Interests	User wants to add all the widgets and filters made by a coworker since it has all the info he needs to track and doesn't want to rebuild it from scratch
Pre-Conditions	Being logged into the platform
Post-Conditions	Dashboard imported into user platform
Main Scenario	
Actor	**System**
1-Creates a new dashboard 2-Selects option to import dashboard 3-Inserts file path	4-Reads file 5-Loads data used in all widgets loaded 6-Creates widgets layout 7-Creates widgets filters and drilldowns 8-Apply current filters to widgets
Alternative Flows	None
Extension Points	5.a Link or file corrupted 5.a.1 Warns user about corruption and doesn't add widget 6.a Can't load data used in the widget from database 6.a.1 Reconnects to database and warns user about wait time bigger than usual 8.a Filters or drilldowns cannot be applied 8.a.1 Creates only the widgets without applying them and sends warning to user

Table A.20: **FR20: Exporting Dashboard**

Use Case 20	Exporting Dashboard
Level	Sea
Priority	Won't Have
Actor	User
Description and Goal	Users can export their current dashboard, along with all the widgets they made, to other users via a link or file
Trigger	Selecting download dashboard

Environment	Dashboard
Stakeholders Interests	User wants to share all widgets, filters and drilldowns created by him to his coworker in order to share the information provided by the dashboard,without having to write down how he built all the widgets
Pre-Conditions	Having a dashboard with at least one widget
Post-Conditions	Creates file to send to other users
Main Scenario	
Actor	**System**
1-Selects options in dashboard 2-Clicks export button	3-Detects what charts, filter and data are used for every widget in the dashboard 4-Writes to and downloads file
Extension Points	None
Alternative Flows	None

Table A.21: **FR21: Email Reports**

Use Case 21	Email Reports
Level	Sea
Priority	Could Have
Actor	User
Trigger	Selecting report option
Stakeholders Interests	User wants to receive updates on their dashboards as a way to be updated constantly on all the dashboards they deem important
Pre-Conditions	Having a dashboard created with widgets
Post-Conditions	Snapshot of dashboard sent to email at the time asked
Main Scenario	
Actor	**System**
1-Selects options and Snapshot Schedule	
	2-Opens popup to user selected date and occurrence
3-Selects what day to start and occurrence	

	4-At the given time and schedule sends email with dashboard snapshot
Alternative Flows	None
Extension Points	4.a Problems while sending email 4.a.1 Report is not sent

Appendix B

Mockups

The rest of the mockups:

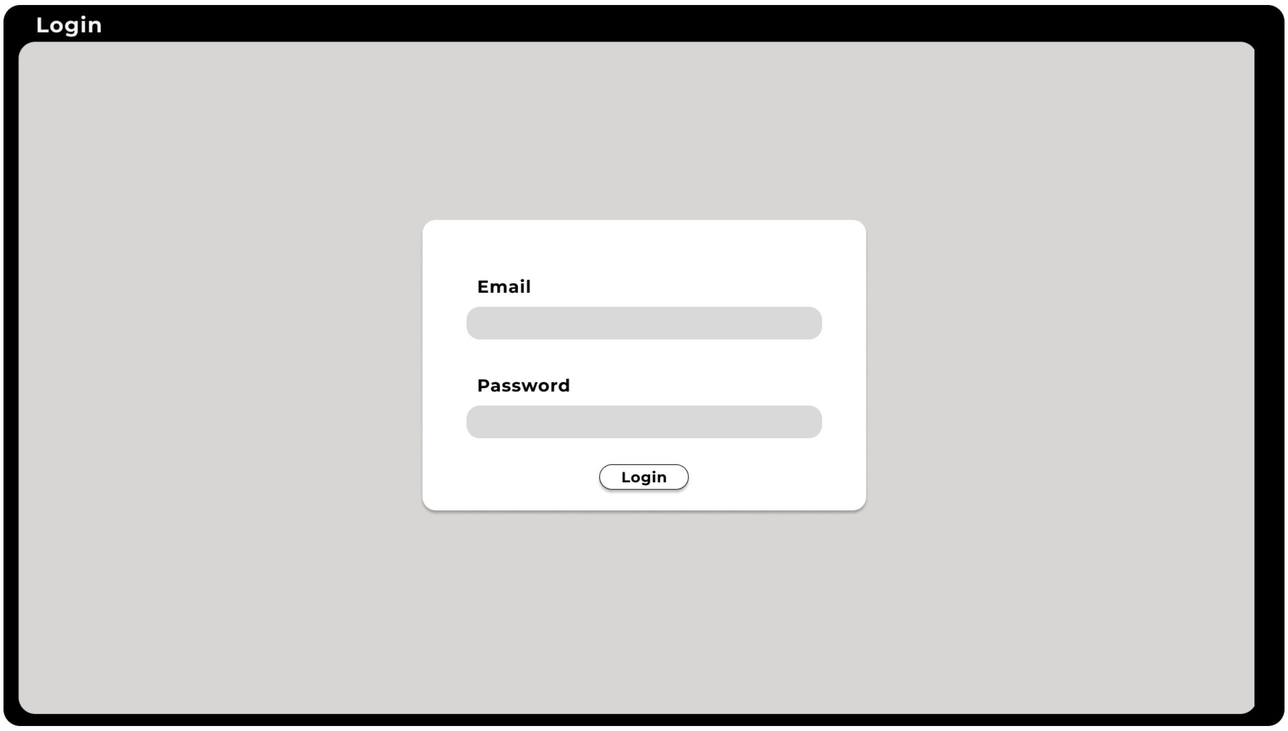


Figure B.1: Login Page

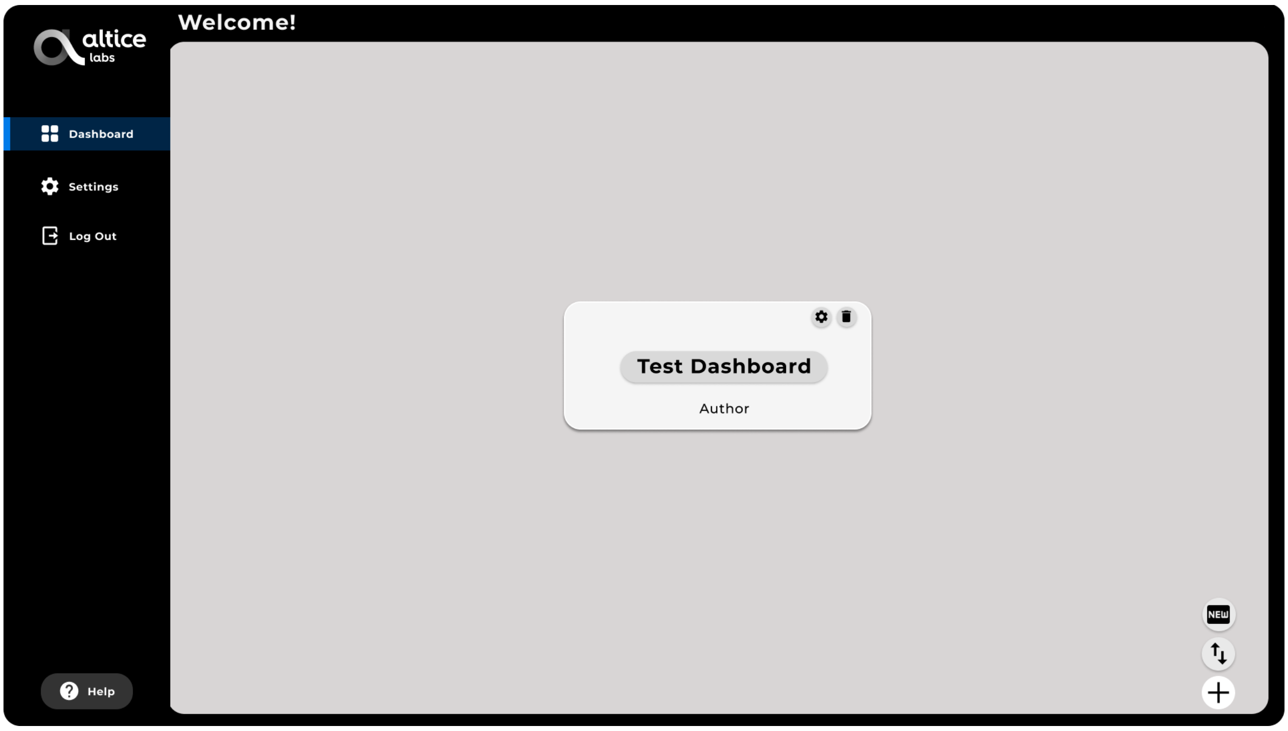


Figure B.2: Navbar on the left

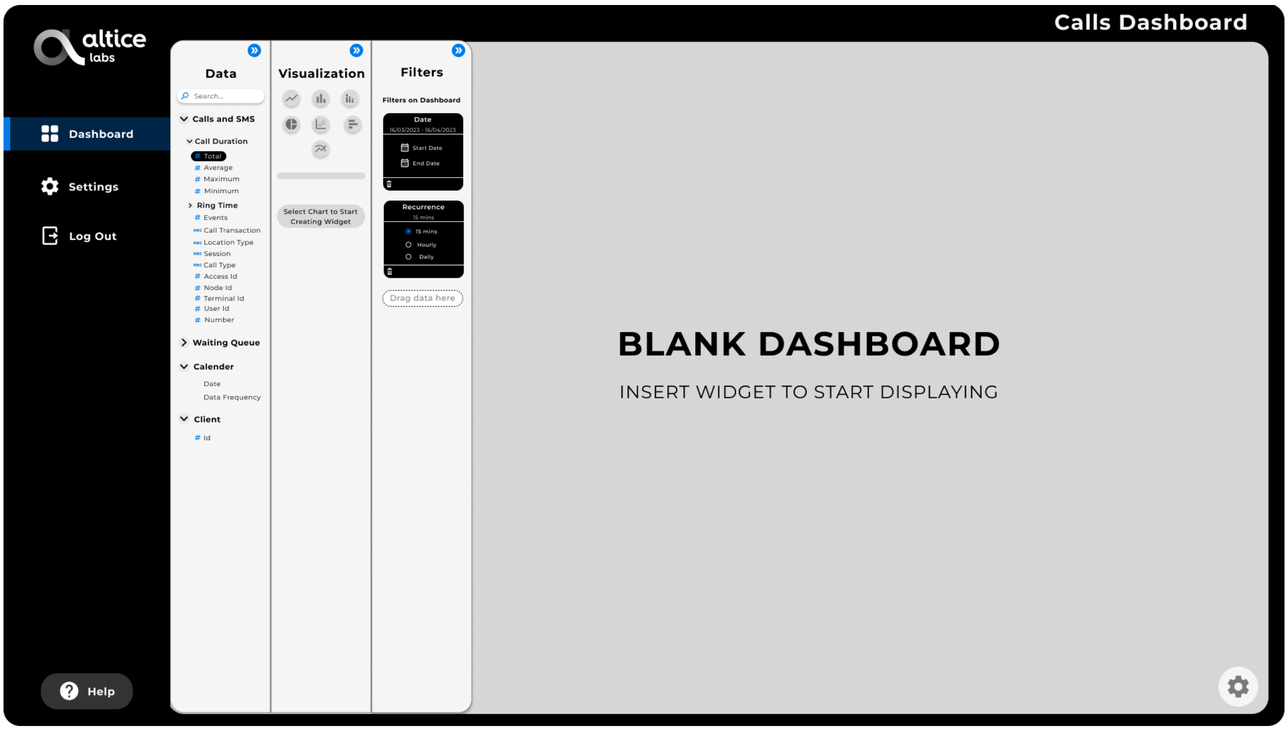


Figure B.3: Dashboard with tabs on the left

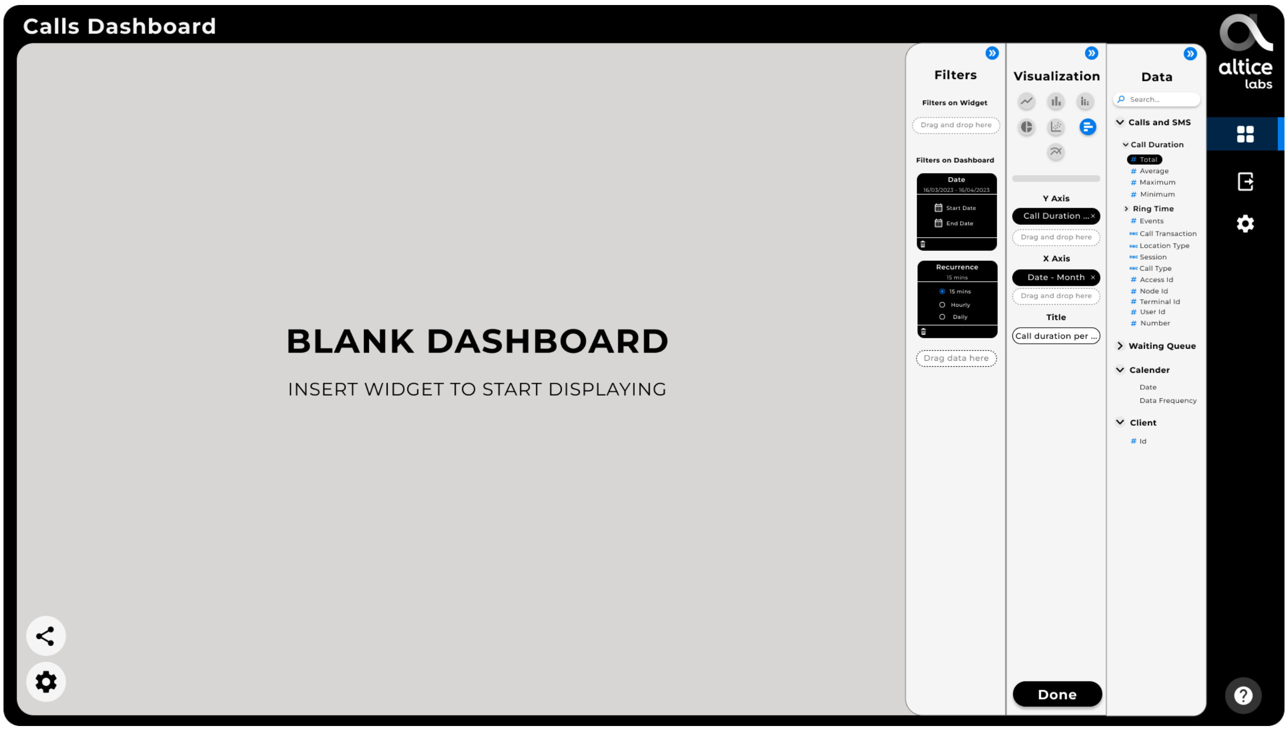


Figure B.4: Creation of an inverted axis widget

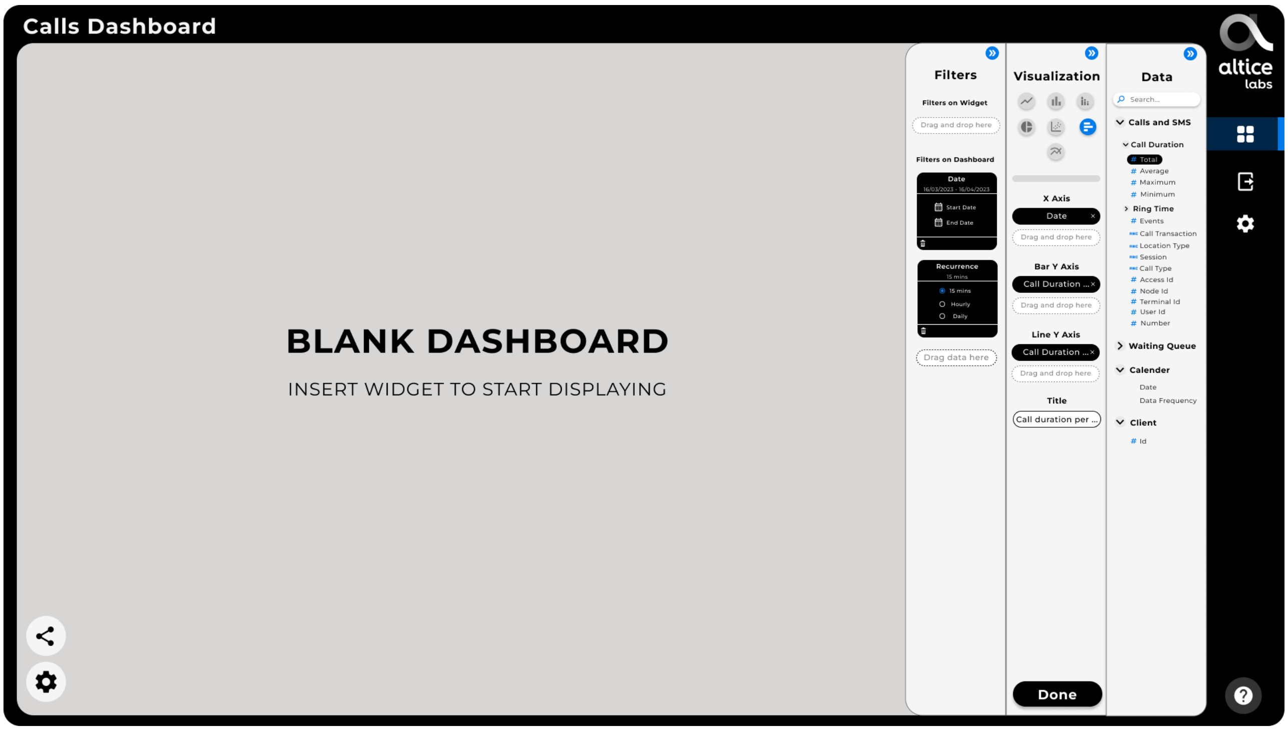
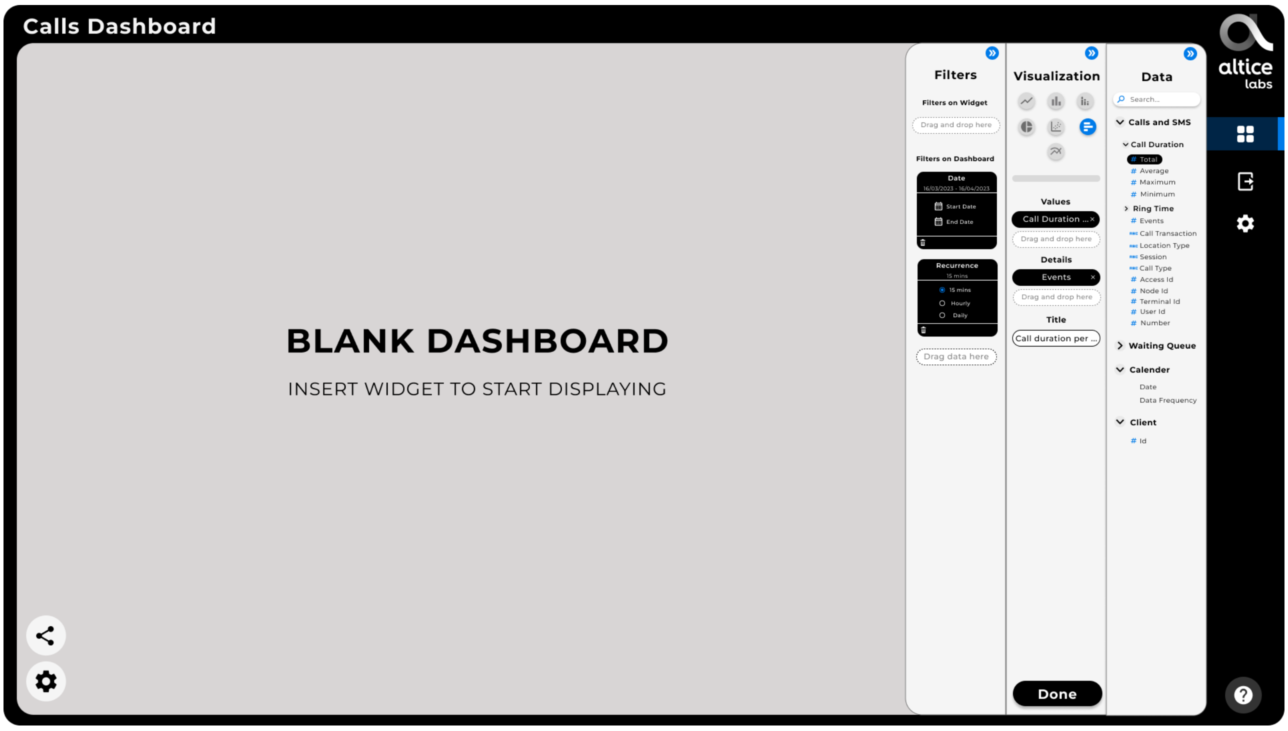


Figure B.5: Creation of a line-column widget



Calls Dashboard

BLANK DASHBOARD

INSERT WIDGET TO START DISPLAYING

Filters

Filters on Widget

Drag and drop here

Filters on Dashboard

Date
16/03/2023 - 16/04/2023
Start Date
End Date

Recurrence
15 mins
[✓ 15 mins]
Hourly
Daily

Drag data here

Visualization

Values
Call Duration ...
Drag and drop here

Details
Events
Drag and drop here

Title
Call duration per ...

Done

Data

Search...

Calls and SMS
- Call Duration
  - # Total
  - # Average
  - # Maximum
  - # Minimum
- Ring Time
  - # Events
  - Call Transaction
  - Location Type
  - Session
  - Call Type
  - # Access Id
  - # Node Id
  - # Terminal Id
  - # User Id
  - # Number

Waiting Queue

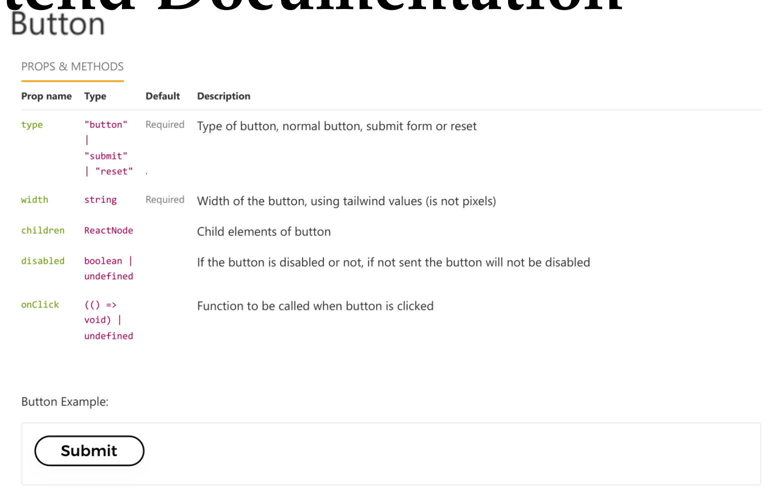
Calender
- Date
- Data Frequency

Client
- # Id

Figure B.6: Creation of a no axis widget

Appendix C

Frontend Documentation

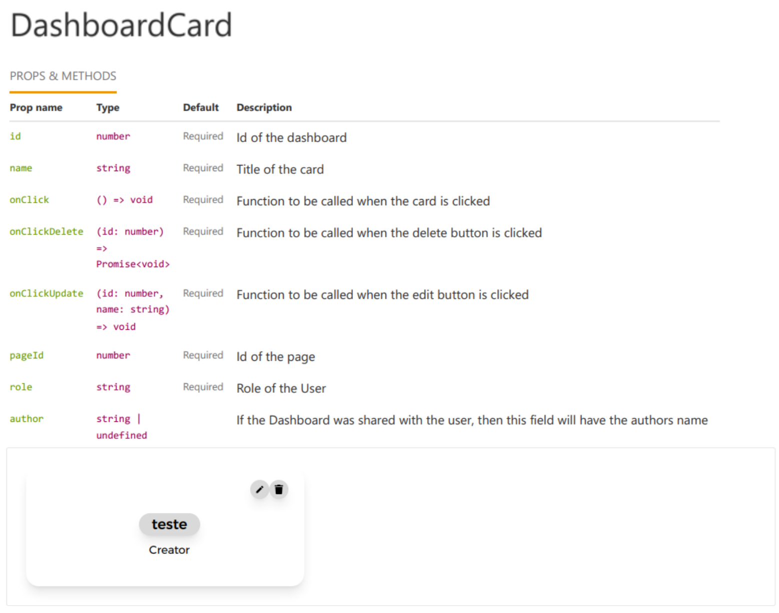


Button

PROPS & METHODS

Prop name	Type	Default	Description
`type`	`"button" | "submit" | "reset"`	Required	Type of button, normal button, submit form or reset
`width`	`string`	Required	Width of the button, using tailwind values (is not pixels)
`children`	`ReactNode`		Child elements of button
`disabled`	`boolean | undefined`		If the button is disabled or not, if not sent the button will not be disabled
`onClick`	`(() => void) | undefined`		Function to be called when button is clicked

Button Example:



DashboardCard

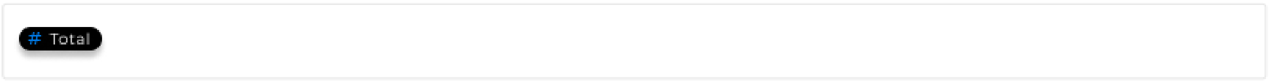
PROPS & METHODS

Prop name	Type	Default	Description
`id`	`number`	Required	Id of the dashboard
`name`	`string`	Required	Title of the card
`onClick`	`() => void`	Required	Function to be called when the card is clicked
`onClickDelete`	`(id: number) => Promise<void>`	Required	Function to be called when the delete button is clicked
`onClickUpdate`	`(id: number, name: string) => void`	Required	Function to be called when the edit button is clicked
`pageId`	`number`	Required	Id of the page
`role`	`string`	Required	Role of the User
`author`	`string | undefined`		If the Dashboard was shared with the user, then this field will have the authors name

Draggable

PROPS & METHODS

Prop name	Type	Default	Description
children	ReactNode	Required	Children of the Draggable
field_id	number	Required	Id of the field that the draggable represents
label	string	Required	Title of the Main Page
nPages	number	Required	Number of pages that the field has in order to display the correct number of pages in the pagination for the filter
name	string	Required	Name of the draggable
onDrop	(item: IField, axis: string, statistic_id: number) => void	Required	Function to be called when the draggable is dragged onto a droppable component, item corresponds to the item dragged, axis to what axis is beeing dragged (x,y or a filter) and statistic_id to the id of the statistic
statistic_id	number	Required	Id of the statistic that the draggable represents
type	string	Required	Type of the draggable
field_values	IFieldValue[] | undefined		In case the field has values they are passed here

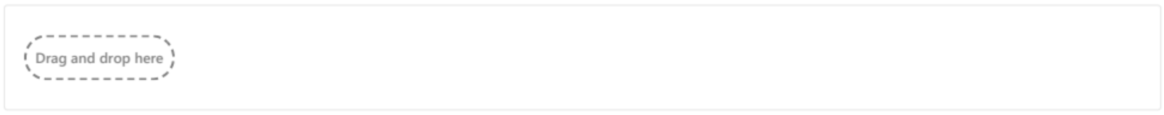


Dropzone

PROPS & METHODS

Prop name	Type	Default	Description
id	string	Required	Id of the dropzone

Dropzone Example:

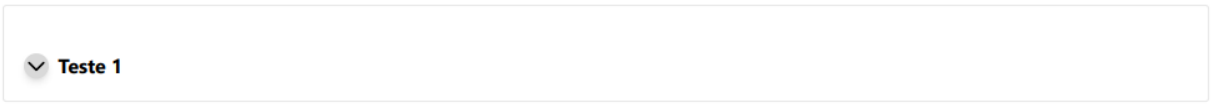


Dropdown

PROPS & METHODS

Prop name	Type	Default	Description
`canBeSelected`	`boolean`	Required	Indicates if the dropdown can be selected or not
`index`	`number`	Required	Index of the dropdown, used to identify which dropdown is being clicked as there can be multiple dropdowns on the same page
`isOpen`	`boolean`	Required	If the dropdown is open or not
`kpi`	`IKPI`	Required	Metrics to be displayed in the dropdown
`name`	`string`	Required	Title of the dropdown
`onClickDropdown`	`(index: number) => void`	Required	Function to be called when the dropdown is clicked
`onDrop`	`(item: IField, axis: string, statistic_id: number) => void`	Required	Function to pass to child components to be called when a metric is dragged onto a droppable component, item corresponds to the item dragged, axis to what axis is beeing dragged (x,y or a filter) and statistic_id to the id of the statistic

Dropdown Example with elements:



Teste 1

FAB

PROPS & METHODS

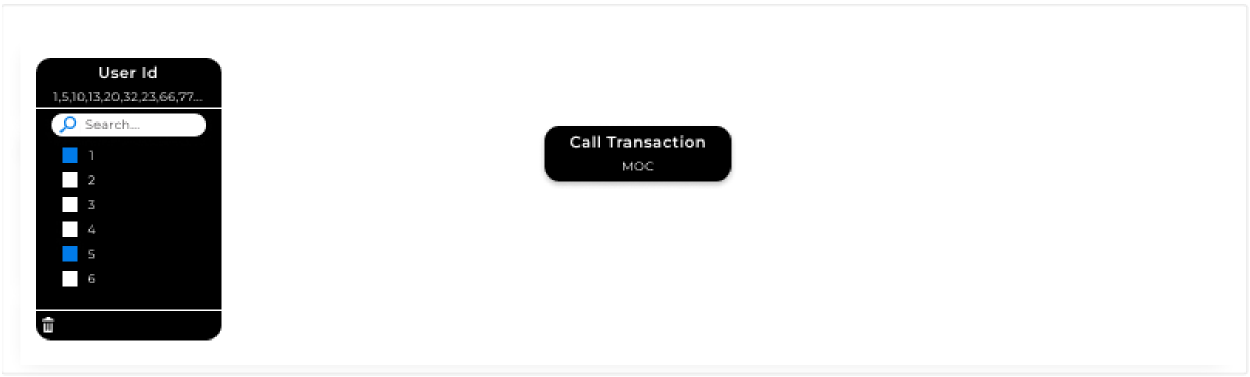
Prop name	Type	Default	Description
`actions`	`{ label: string; onClick: () => void; icon: Element; }[]`	Required	All the icons that are supposed to be displayed in the FAB
`settings`	`boolean`	Required	If the FAB is being used in the settings or not (needed as the style has to be a bit different in order for the FAB to be displayed in the correct box)
`side`	`string`	Required	Side of the screen the FAB is supposed to be on
`documentation`	`boolean | undefined`		If the FAB is being used in the documentation or not (needed as the style has to be a bit different in order for the FAB to be displayed in the correct box)

FAB Example:

FilterCard

PROPS & METHODS

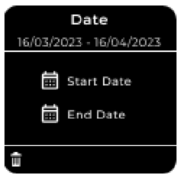
Prop name	Type	Default	Description
`id`	`number`	Required	Id of the field on the filter
`index`	`number`	Required	Index of the filter, used to identify which filter is being clicked as there can be multiple filters on the same page
`isGlobal`	`boolean`	Required	If the filter is global or not
`nPages`	`number`	Required	Number of pages that the field has in order to display the correct number of pages in the pagination for the filter
`onClick`	`(isGlobal: boolean, index: number) => void`	Required	When the filter is clicked it will open the filter modal or close it if it is already open
`recurrence`	`string`	Required	Recurrence of the data
`selected`	`string[]`	Required	Array with the selected values of the filter
`statisticId`	`number`	Required	Id of the statistic that the filter is being applied to
`title`	`string`	Required	Title of the FilterCard
`addValueToFilter`	`((indexFilter: number, value: string, isGlobal: boolean) => void) | undefined`		Function to be called when the user selects value in filter
`label`	`string | undefined`		Label for the subtitle of the filter card
`onTrash`	`((isGlobal: boolean, index: number) => void) | undefined`		Function to be called when the trash button is clicked, it will delete the filter
`removeValueToFilter`	`((indexFilter: number, value: string, isGlobal: boolean) =>`		Function to be called when the user removes value in filter



FilterCardDate

PROPS & METHODS

Prop name	Type	Default	Description
`endDate`	`Date`	Required	End date selected
`onClick`	`(isGlobal: boolean, index: number) => void`	Required	When the filter is clicked it will open the filter modal or close it if it is already open
`onDateChange`	`(startDate: boolean, date: Date) => void`	Required	Function called when user selects a different date
`startDate`	`Date`	Required	Start date selected

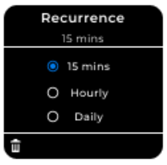


FilterCardRecurrence

PROPS & METHODS

Prop name	Type	Default	Description
`onClick`	`(isGlobal: boolean, index: number) => void`	Required	When the filter is clicked it will open the filter modal or close it if it is already open
`onRecurrenceChange`	`(option: string) => void`	Required	Function called when user selects a different recurrence
`selected`	`string`	Required	Recurrence selected

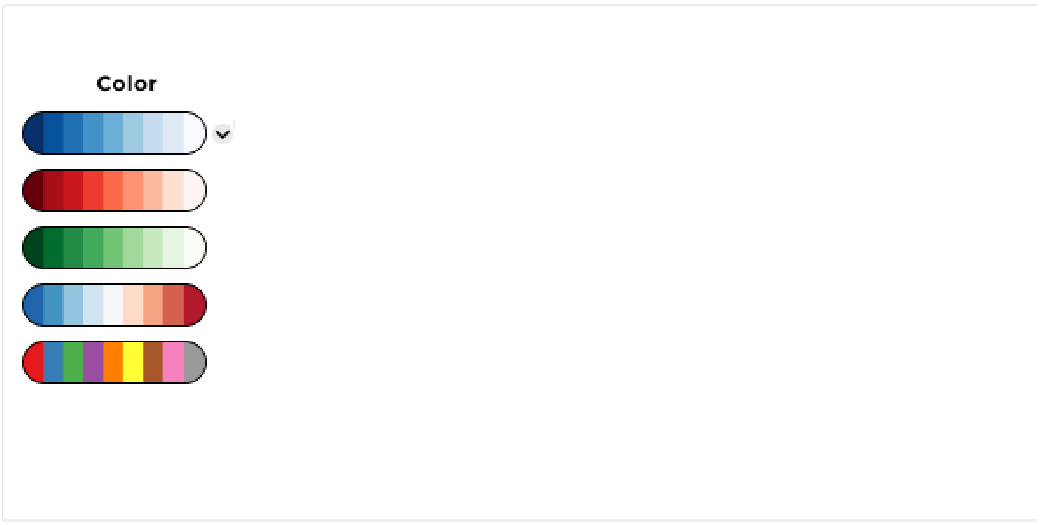
Filter Card Recurre Example:



PalettePicker

PROPS & METHODS

Prop name	Type	Default	Description
`isOpened`	`boolean`	Required	If palette dropdown is opened or not
`onClickArrow`	`() => void`	Required	Function to be called when user opnes/closes dropdown
`onClickRange`	`(index: number) => void`	Required	Function to be called when user clicks on new range of colors
`ranges`	`IRange[]`	Required	List of all range of colors available
`selected`	`number`	Required	Index of color selected
`isDocumentation`	`boolean | undefined`		If it is documentation then styles have to be different

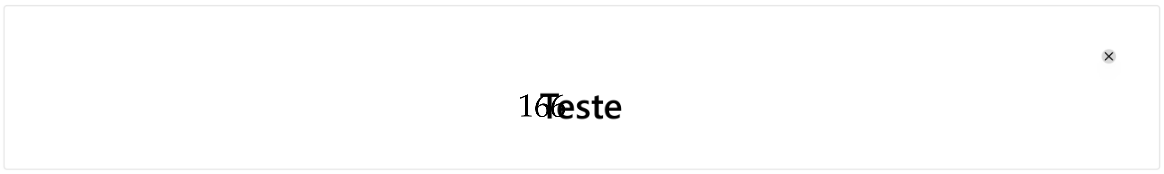


PopUp

PROPS & METHODS

Prop name	Type	Default	Description
`children`	`ReactNode`	Required	Children of the PopUp
`height`	`string`	Required	Height of the PopUp in pixels
`onClickOut`	`() => void`	Required	Function to be called when clicked out of the PopUp
`title`	`string`	Required	Title of the PopUp
`width`	`string`	Required	Width of the PopUp in pixels

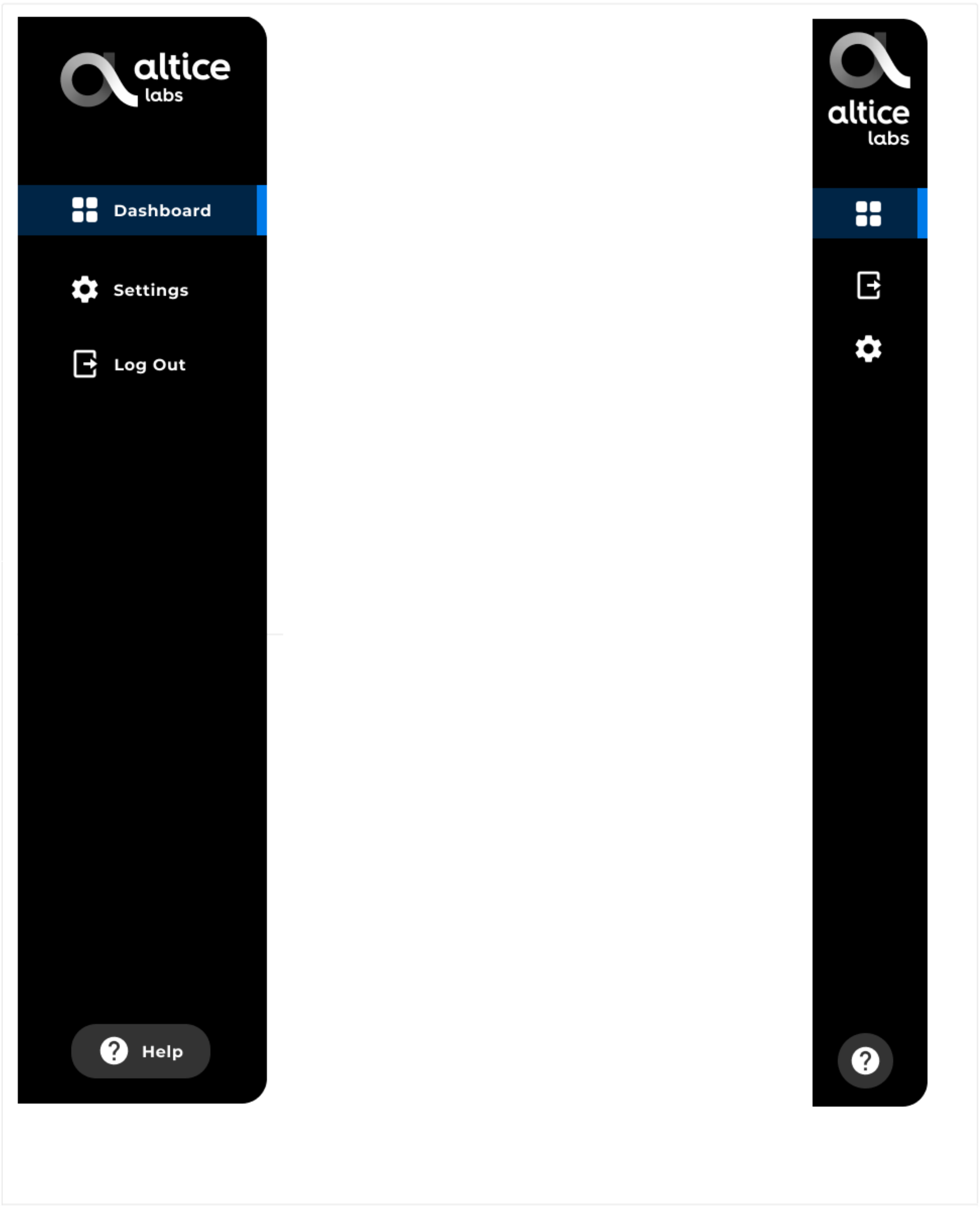
PopUp Example:



SideBar

PROPS & METHODS

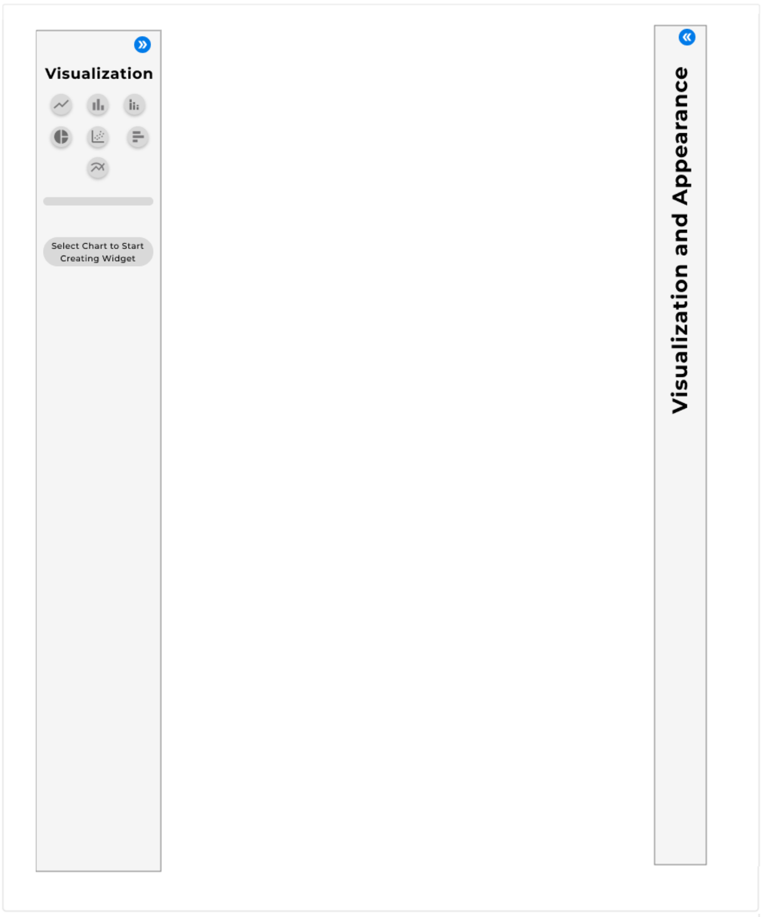
Prop name	Type	Default	Description
`setIsOpen`	`Dispatch<SetStateAction<boolean>>`	Required	Function to close or open the sidebar
`setTitleDashboard`	`() => void`	Required	Change title of the page
`side`	`string`	Required	Side of the Sidebar, left or right
`isDocumentation`	`boolean | undefined`		If it is the settings page or not



ToolBar

PROPS & METHODS

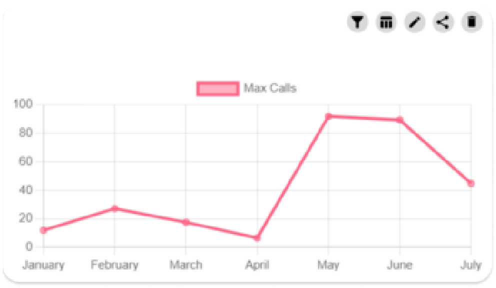
Prop name	Type	Default	Description
`children`	`ReactNode`	Required	Children of the ToolBar
`setIsOpen`	`(toolbarNumber: number) => void`	Required	Open or close the ToolBar
`style`	`string`	Required	Style of the ToolBar, using tailwind values
`toolbarNumber`	`number`	Required	Identifies the ToolBar
`documentation`	`boolean | undefined`		If the ToolBar is being used in the documentation or not (needed as the style has to be a bit different in order for the ToolBar to be displayed in the correct box)


Visualization
Select Chart to Start Creating Widget
Visualization and Appearance

WidgetCard

PROPS & METHODS

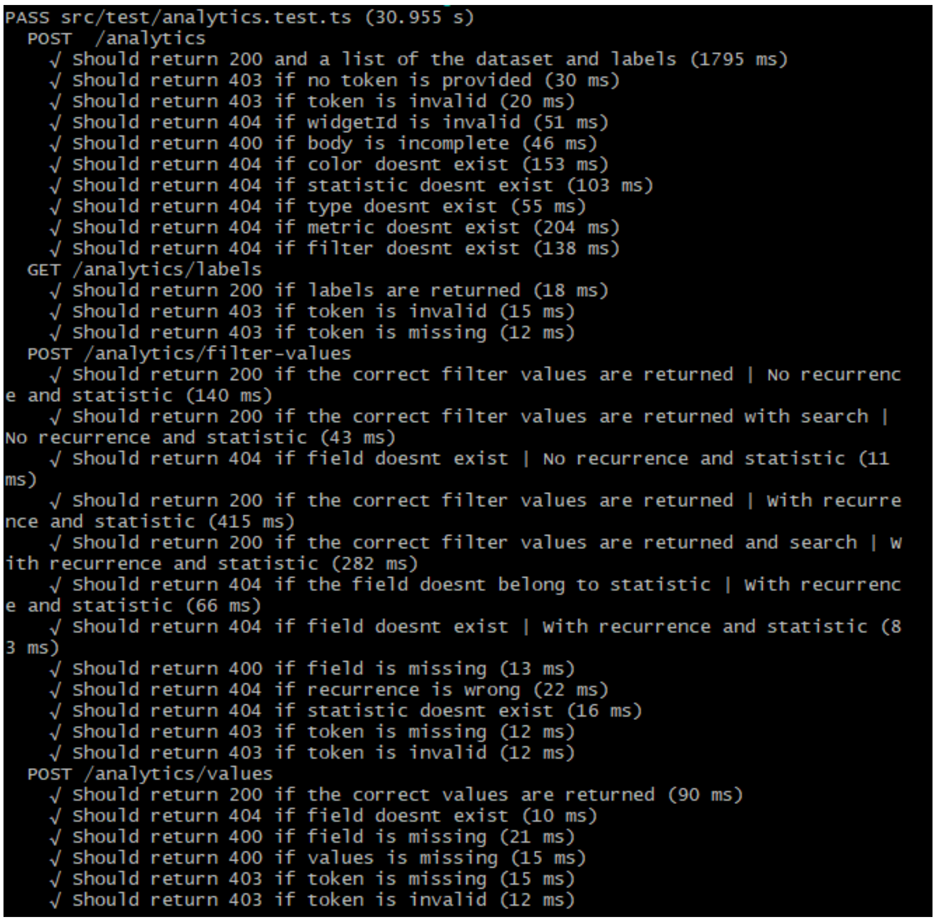
Prop name	Type	Default	Description
`children`	`ReactNode`	Required	Children of the widgetcard, in this case the chart
`index`	`number`	Required	Index in the array of widgets
`isClicked`	`boolean`	Required	If its clicked then the widgetcard will pop to the top of page (bigger z-axis)
`isViewer`	`boolean`	Required	If is viewer or not, if its it cant be dragged or resized
`onClick`	`(index: number) => void`	Required	Function to be called when the widgetcard is clicked, will pop the widgetcard to the top of the page, enable widget filtering and edition
`onDragStop`	`(index: number, position: { x: number; y: number; }) => void`	Required	Function to be called when the user drags it will update the position of the widgetcard
`onEdit`	`(index: number) => void`	Required	Function to be called when the user clicks the edit button, will open the edit tab
`onFilter`	`(index: number) => void`	Required	Function to be called when the user clicks the filter button, will open the filter tab
`onResizeStop`	`(index: number, style: { width: string; height: string; }) => void`	Required	Function to be called when the user resizes it will update the size of the widgetcard
`onShare`	`(index: number) => void`	Required	Function to be called when the user clicks the share button, will open the share page
`onTable`	`(index: number) => void`	Required	Function to be called when the user clicks the table button, will open the table tab
`onTrash`	`(index: number, event?: MouseEvent<HTMLDivElement, MouseEvent> | undefined) => void`	Required	Deletes the widget when the trash button is clicked
`position`	`{ x: number; y: number; }`	Required	Position of the WidgetCard
`size`	`{ width: number; height: number; }`	Required	Size of the WidgetCard
`title`	`string`	Required	Title of the WidgetCard



This page is intentionally left blank.

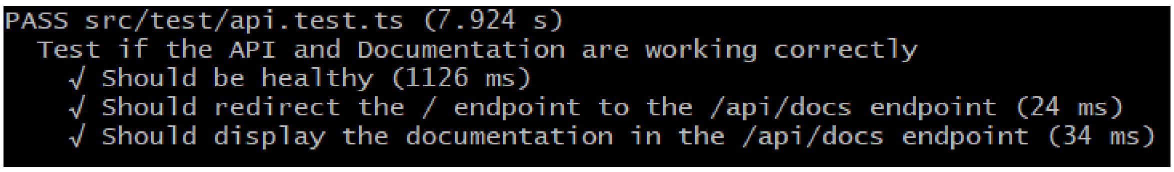
Appendix D

API Testing



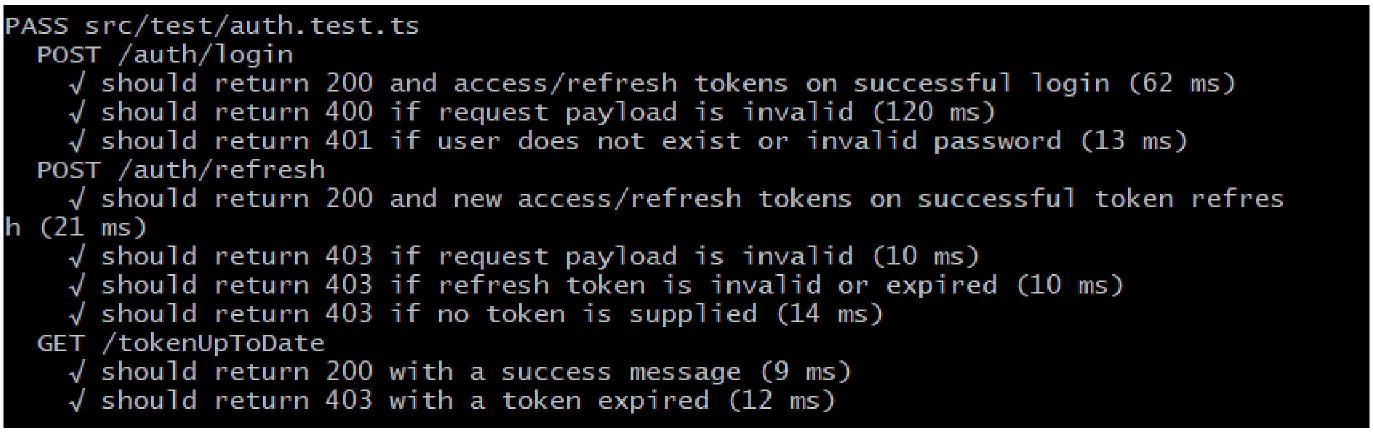
```
PASS src/test/analytics.test.ts (30.955 s)
  POST  /analytics
    √ Should return 200 and a list of the dataset and labels (1795 ms)
    √ Should return 403 if no token is provided (30 ms)
    √ Should return 403 if token is invalid (20 ms)
    √ Should return 404 if widgetId is invalid (51 ms)
    √ Should return 400 if body is incomplete (46 ms)
    √ Should return 404 if color doesnt exist (153 ms)
    √ Should return 404 if statistic doesnt exist (103 ms)
    √ Should return 404 if type doesnt exist (55 ms)
    √ Should return 404 if metric doesnt exist (204 ms)
    √ Should return 404 if filter doesnt exist (138 ms)
  GET /analytics/labels
    √ Should return 200 if labels are returned (18 ms)
    √ Should return 403 if token is invalid (15 ms)
    √ Should return 403 if token is missing (12 ms)
  POST /analytics/filter-values
    √ Should return 200 if the correct filter values are returned | No recurrenc
e and statistic (140 ms)
    √ Should return 200 if the correct filter values are returned with search |
No recurrence and statistic (43 ms)
    √ Should return 404 if field doesnt exist | No recurrence and statistic (11
ms)
    √ Should return 200 if the correct filter values are returned | With recurre
nce and statistic (415 ms)
    √ Should return 200 if the correct filter values are returned and search | W
ith recurrence and statistic (282 ms)
    √ Should return 404 if the field doesnt belong to statistic | With recurrenc
e and statistic (66 ms)
    √ Should return 404 if field doesnt exist | With recurrence and statistic (8
3 ms)
    √ Should return 400 if field is missing (13 ms)
    √ Should return 404 if recurrence is wrong (22 ms)
    √ Should return 404 if statistic doesnt exist (16 ms)
    √ Should return 403 if token is missing (12 ms)
    √ Should return 403 if token is invalid (12 ms)
  POST /analytics/values
    √ Should return 200 if the correct values are returned (90 ms)
    √ Should return 404 if field doesnt exist (10 ms)
    √ Should return 400 if field is missing (21 ms)
    √ Should return 400 if values is missing (15 ms)
    √ Should return 403 if token is missing (15 ms)
    √ Should return 403 if token is invalid (12 ms)
```

Figure D.1: Tests done to the analytics endpoints



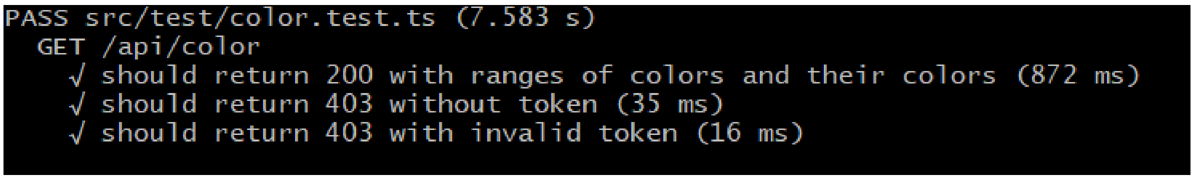
```
PASS src/test/api.test.ts (7.924 s)
  Test if the API and Documentation are working correctly
    √ Should be healthy (1126 ms)
    √ Should redirect the / endpoint to the /api/docs endpoint (24 ms)
    √ Should display the documentation in the /api/docs endpoint (34 ms)
```

Figure D.2: Tests done to the API health endpoint



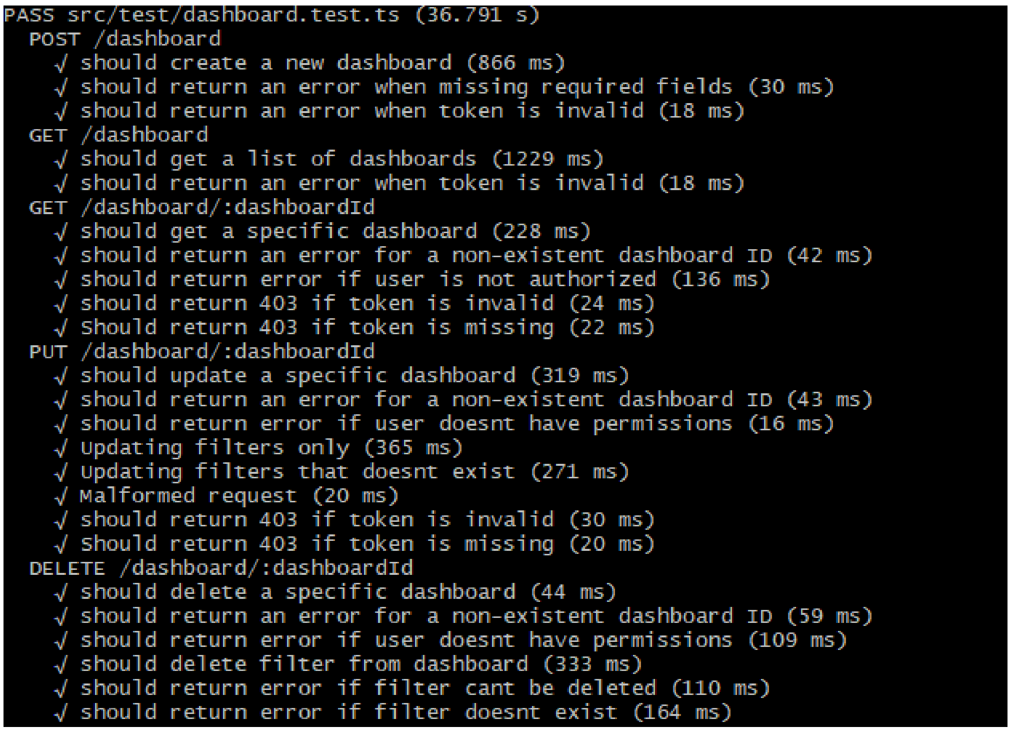
```
PASS src/test/auth.test.ts
  POST /auth/login
    √ should return 200 and access/refresh tokens on successful login (62 ms)
    √ should return 400 if request payload is invalid (120 ms)
    √ should return 401 if user does not exist or invalid password (13 ms)
  POST /auth/refresh
    √ should return 200 and new access/refresh tokens on successful token refres
h (21 ms)
    √ should return 403 if request payload is invalid (10 ms)
    √ should return 403 if refresh token is invalid or expired (10 ms)
    √ should return 403 if no token is supplied (14 ms)
  GET /tokenUpToDate
    √ should return 200 with a success message (9 ms)
    √ should return 403 with a token expired (12 ms)
```

Figure D.3: Tests done to the authorization endpoint



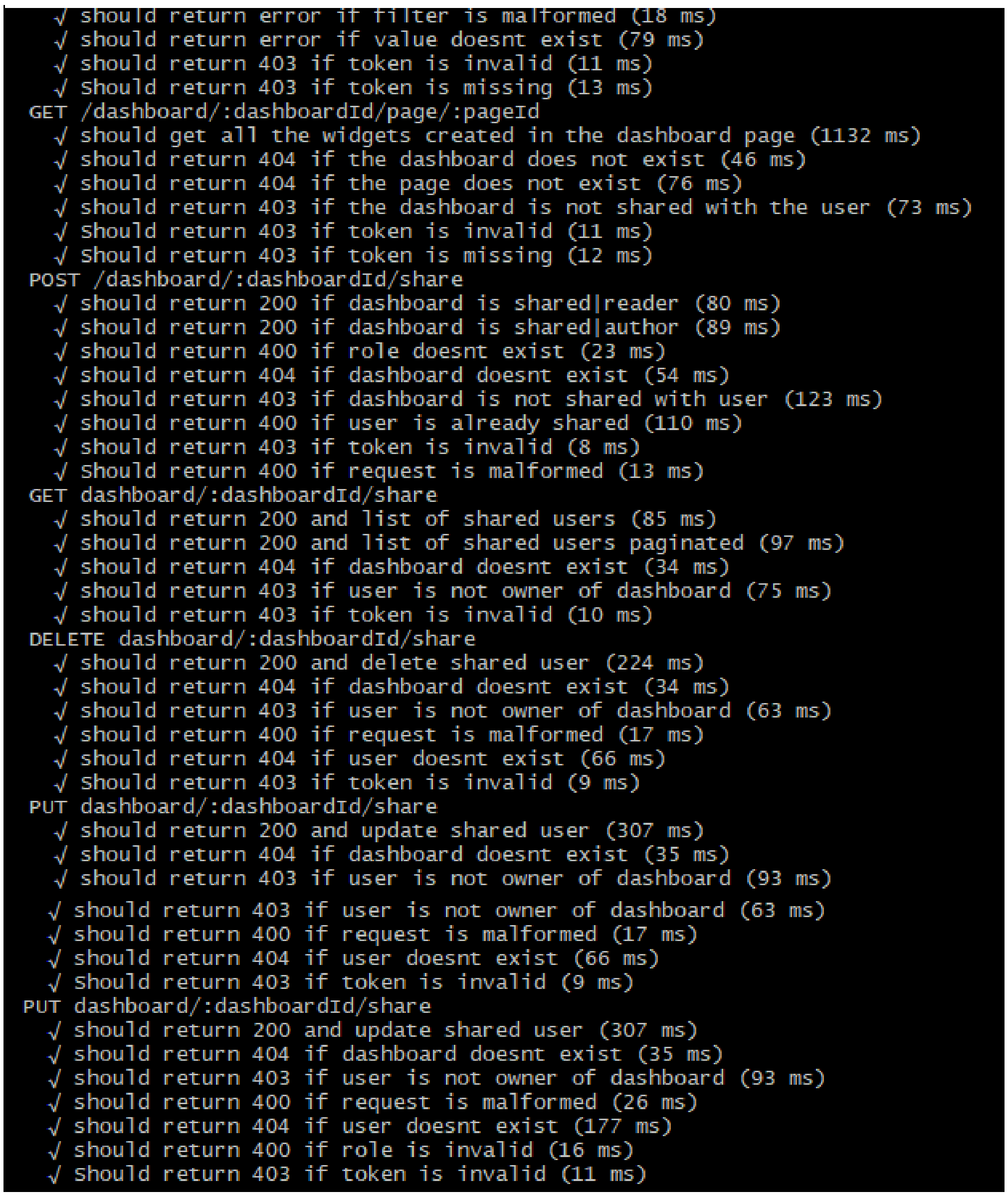
```
PASS src/test/color.test.ts (7.583 s)
  GET /api/color
    √ should return 200 with ranges of colors and their colors (872 ms)
    √ should return 403 without token (35 ms)
    √ should return 403 with invalid token (16 ms)
```

Figure D.4: Tests done to the color endpoint



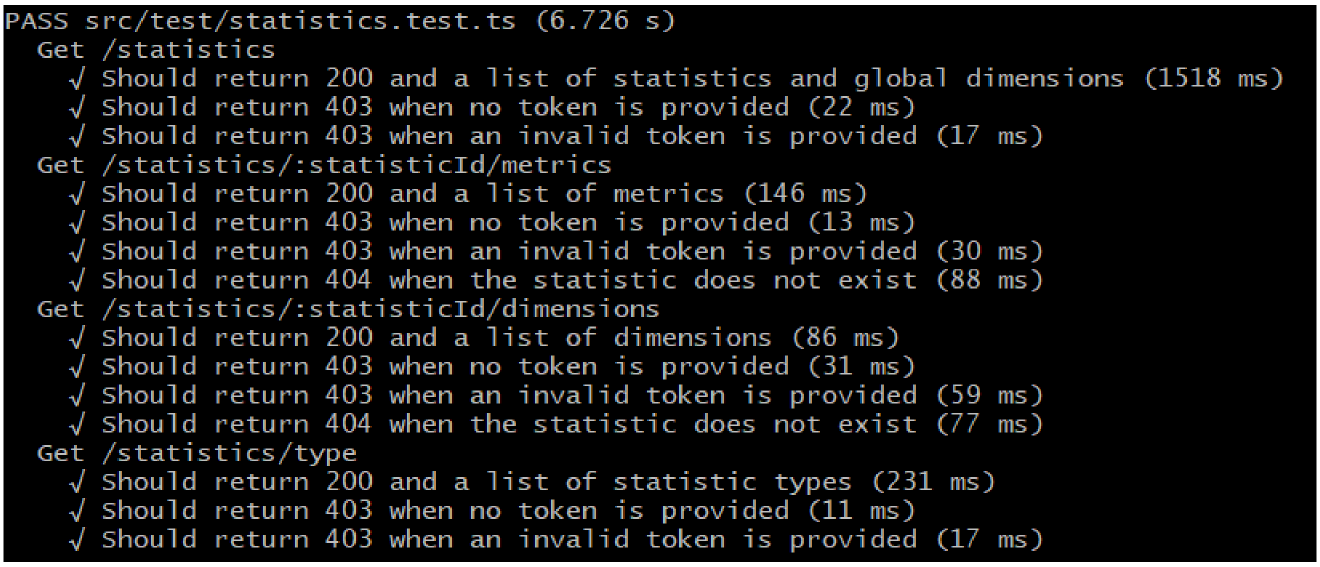
```
PASS src/test/dashboard.test.ts (36.791 s)
  POST /dashboard
    √ should create a new dashboard (866 ms)
    √ should return an error when missing required fields (30 ms)
    √ should return an error when token is invalid (18 ms)
  GET /dashboard
    √ should get a list of dashboards (1229 ms)
    √ should return an error when token is invalid (18 ms)
  GET /dashboard/:dashboardId
    √ should get a specific dashboard (228 ms)
    √ should return an error for a non-existent dashboard ID (42 ms)
    √ should return error if user is not authorized (136 ms)
    √ should return 403 if token is invalid (24 ms)
    √ Should return 403 if token is missing (22 ms)
  PUT /dashboard/:dashboardId
    √ should update a specific dashboard (319 ms)
    √ should return an error for a non-existent dashboard ID (43 ms)
    √ should return error if user doesnt have permissions (16 ms)
    √ Updating filters only (365 ms)
    √ Updating filters that doesnt exist (271 ms)
    √ Malformed request (20 ms)
    √ should return 403 if token is invalid (30 ms)
    √ Should return 403 if token is missing (20 ms)
  DELETE /dashboard/:dashboardId
    √ should delete a specific dashboard (44 ms)
    √ should return an error for a non-existent dashboard ID (59 ms)
    √ should return error if user doesnt have permissions (109 ms)
    √ should delete filter from dashboard (333 ms)
    √ should return error if filter cant be deleted (110 ms)
    √ should return error if filter doesnt exist (164 ms)
```

Figure D.5: Tests done to the authorization endpoint - First Part



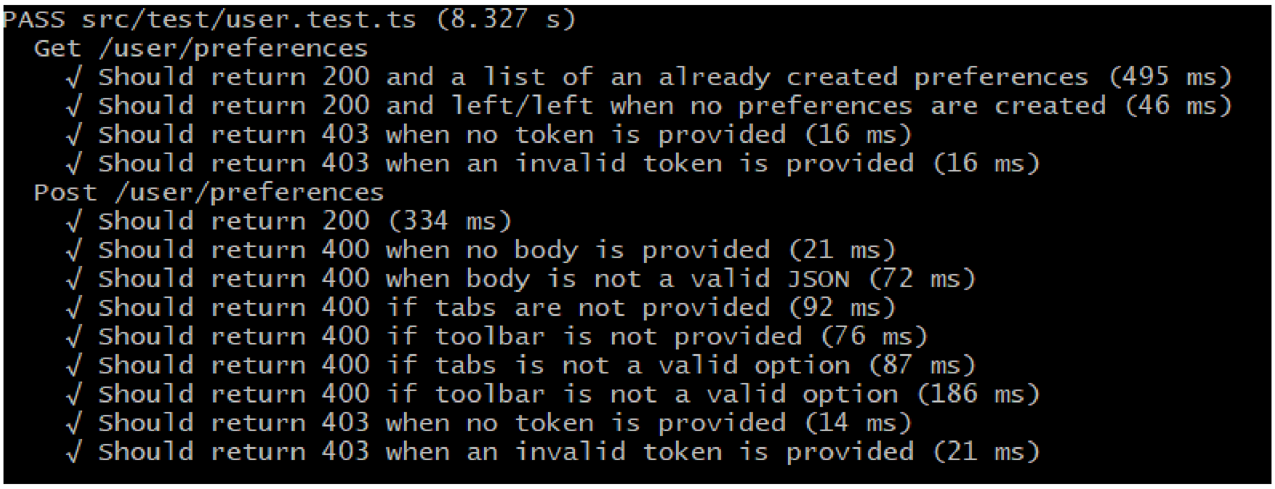
```
  √ should return error if filter is malformed (18 ms)
  √ should return error if value doesnt exist (79 ms)
  √ should return 403 if token is invalid (11 ms)
  √ Should return 403 if token is missing (13 ms)
GET /dashboard/:dashboardId/page/:pageId
  √ should get all the widgets created in the dashboard page (1132 ms)
  √ should return 404 if the dashboard does not exist (46 ms)
  √ should return 404 if the page does not exist (76 ms)
  √ should return 403 if the dashboard is not shared with the user (73 ms)
  √ Should return 403 if token is invalid (11 ms)
  √ Should return 403 if token is missing (12 ms)
POST /dashboard/:dashboardId/share
  √ should return 200 if dashboard is shared|reader (80 ms)
  √ should return 200 if dashboard is shared|author (89 ms)
  √ should return 400 if role doesnt exist (23 ms)
  √ should return 404 if dashboard doesnt exist (54 ms)
  √ should return 403 if dashboard is not shared with user (123 ms)
  √ should return 400 if user is already shared (110 ms)
  √ should return 403 if token is invalid (8 ms)
  √ Should return 400 if request is malformed (13 ms)
GET dashboard/:dashboardId/share
  √ should return 200 and list of shared users (85 ms)
  √ should return 200 and list of shared users paginated (97 ms)
  √ should return 404 if dashboard doesnt exist (34 ms)
  √ should return 403 if user is not owner of dashboard (75 ms)
  √ should return 403 if token is invalid (10 ms)
DELETE dashboard/:dashboardId/share
  √ should return 200 and delete shared user (224 ms)
  √ should return 404 if dashboard doesnt exist (34 ms)
  √ should return 403 if user is not owner of dashboard (63 ms)
  √ should return 400 if request is malformed (17 ms)
  √ should return 404 if user doesnt exist (66 ms)
  √ Should return 403 if token is invalid (9 ms)
PUT dashboard/:dashboardId/share
  √ should return 200 and update shared user (307 ms)
  √ should return 404 if dashboard doesnt exist (35 ms)
  √ should return 403 if user is not owner of dashboard (93 ms)
  √ should return 403 if user is not owner of dashboard (63 ms)
  √ should return 400 if request is malformed (17 ms)
  √ should return 404 if user doesnt exist (66 ms)
  √ Should return 403 if token is invalid (9 ms)
PUT dashboard/:dashboardId/share
  √ should return 200 and update shared user (307 ms)
  √ should return 404 if dashboard doesnt exist (35 ms)
  √ should return 403 if user is not owner of dashboard (93 ms)
  √ should return 400 if request is malformed (26 ms)
  √ should return 404 if user doesnt exist (177 ms)
  √ should return 400 if role is invalid (16 ms)
  √ Should return 403 if token is invalid (11 ms)
```

Figure D.6: Tests done to the authorization endpoint - Second Part



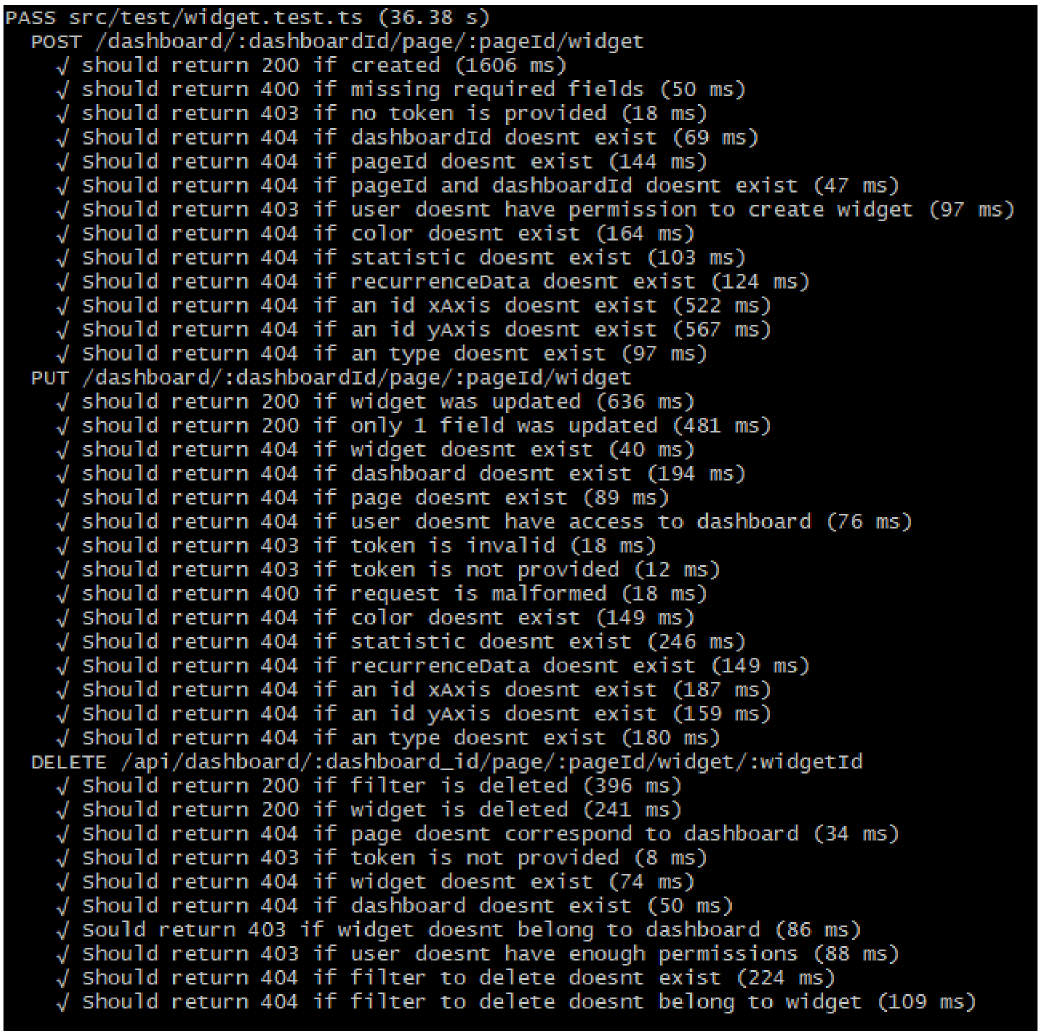
```
PASS src/test/statistics.test.ts (6.726 s)
  Get /statistics
    √ Should return 200 and a list of statistics and global dimensions (1518 ms)
    √ Should return 403 when no token is provided (22 ms)
    √ Should return 403 when an invalid token is provided (17 ms)
  Get /statistics/:statisticId/metrics
    √ Should return 200 and a list of metrics (146 ms)
    √ Should return 403 when no token is provided (13 ms)
    √ Should return 403 when an invalid token is provided (30 ms)
    √ Should return 404 when the statistic does not exist (88 ms)
  Get /statistics/:statisticId/dimensions
    √ Should return 200 and a list of dimensions (86 ms)
    √ Should return 403 when no token is provided (31 ms)
    √ Should return 403 when an invalid token is provided (59 ms)
    √ Should return 404 when the statistic does not exist (77 ms)
  Get /statistics/type
    √ Should return 200 and a list of statistic types (231 ms)
    √ Should return 403 when no token is provided (11 ms)
    √ Should return 403 when an invalid token is provided (17 ms)
```

Figure D.7: Tests done to the statistic endpoint



```
PASS src/test/user.test.ts (8.327 s)
  Get /user/preferences
    √ Should return 200 and a list of an already created preferences (495 ms)
    √ Should return 200 and left/left when no preferences are created (46 ms)
    √ Should return 403 when no token is provided (16 ms)
    √ Should return 403 when an invalid token is provided (16 ms)
  Post /user/preferences
    √ Should return 200 (334 ms)
    √ Should return 400 when no body is provided (21 ms)
    √ Should return 400 when body is not a valid JSON (72 ms)
    √ Should return 400 if tabs are not provided (92 ms)
    √ Should return 400 if toolbar is not provided (76 ms)
    √ Should return 400 if tabs is not a valid option (87 ms)
    √ Should return 400 if toolbar is not a valid option (186 ms)
    √ Should return 403 when no token is provided (14 ms)
    √ Should return 403 when an invalid token is provided (21 ms)
```

Figure D.8: Tests done to the user endpoint



```
PASS src/test/widget.test.ts (36.38 s)
  POST /dashboard/:dashboardId/page/:pageId/widget
    √ should return 200 if created (1606 ms)
    √ should return 400 if missing required fields (50 ms)
    √ should return 403 if no token is provided (18 ms)
    √ Should return 404 if dashboardId doesnt exist (69 ms)
    √ Should return 404 if pageId doesnt exist (144 ms)
    √ Should return 404 if pageId and dashboardId doesnt exist (47 ms)
    √ Should return 403 if user doesnt have permission to create widget (97 ms)
    √ Should return 404 if color doesnt exist (164 ms)
    √ Should return 404 if statistic doesnt exist (103 ms)
    √ Should return 404 if recurrenceData doesnt exist (124 ms)
    √ Should return 404 if an id xAxis doesnt exist (522 ms)
    √ Should return 404 if an id yAxis doesnt exist (567 ms)
    √ Should return 404 if an type doesnt exist (97 ms)
  PUT /dashboard/:dashboardId/page/:pageId/widget
    √ should return 200 if widget was updated (636 ms)
    √ should return 200 if only 1 field was updated (481 ms)
    √ should return 404 if widget doesnt exist (40 ms)
    √ should return 404 if dashboard doesnt exist (194 ms)
    √ should return 404 if page doesnt exist (89 ms)
    √ should return 404 if user doesnt have access to dashboard (76 ms)
    √ should return 403 if token is invalid (18 ms)
    √ should return 403 if token is not provided (12 ms)
    √ should return 400 if request is malformed (18 ms)
    √ Should return 404 if color doesnt exist (149 ms)
    √ Should return 404 if statistic doesnt exist (246 ms)
    √ Should return 404 if recurrenceData doesnt exist (149 ms)
    √ Should return 404 if an id xAxis doesnt exist (187 ms)
    √ Should return 404 if an id yAxis doesnt exist (159 ms)
    √ Should return 404 if an type doesnt exist (180 ms)
  DELETE /api/dashboard/:dashboard_id/page/:pageId/widget/:widgetId
    √ Should return 200 if filter is deleted (396 ms)
    √ Should return 200 if widget is deleted (241 ms)
    √ Should return 404 if page doesnt correspond to dashboard (34 ms)
    √ Should return 403 if token is not provided (8 ms)
    √ Should return 404 if widget doesnt exist (74 ms)
    √ Should return 404 if dashboard doesnt exist (50 ms)
    √ Sould return 403 if widget doesnt belong to dashboard (86 ms)
    √ Should return 403 if user doesnt have enough permissions (88 ms)
    √ Should return 404 if filter to delete doesnt exist (224 ms)
    √ Should return 404 if filter to delete doesnt belong to widget (109 ms)
```

Figure D.9: Tests done to the widget endpoint

This page is intentionally left blank.

Appendix E

Frontend Testing

Test Case #	Name	Steps	Expected Result	Result
1	Create Dashboard	1. Open Web App 2. Login 3. Hover the "+" button 4. Click on the "add" button 5. Insert dashboard name "Dashboard Test" 6. Click on the submit button	Creates the new dashboard "Dashboard Test" and redirects to the dashboard page	Pass
2	Create Dashboard | Empty name	1. Open Web App 2. Login 3. Hover the "+" button 4. Click on the "add" button 5. Insert blank dashboard name 6. Click on the submit button	Warns user that name cannot be empty	Pass
3	Edit Dashboard	1. Open Web App 2. Login 3. Click on Dashboard "Edit" button 4. Insert dashboard new name "Dashboard Edit" 6. Click on submit button	Changes dashboard name to "Dashboard Edit" but stays on the main page	Pass
4	Edit Dashboard | Empty Name	1. Open Web App 2. Login 3. Click on Dashboard "Edit" button 4. Insert blank dashboard name 6. Click on submit button	Warns user that name cannot be empty	Pass

Test Case #	Name	Steps	Expected Result	Result
5	Delete Dashboard	1. Open Web App 2. Login 3. Click on Dashboard "Delete" button 4. Confirm Deletion	Deletes dashboard and stays on the main page	Pass
6	Open Dashboard | Author	1. Open Web App 2. Login 3. Click on Dashboard name	Redirect to dashboard page where the user will have all the permissions to create/edit/filter widgets and share the dashboard	Pass
7	Open Dashboard | Reader	1. Open Web App 2. Login 3. Click on Dashboard name	Redirect to dashboard page where the user will have all the permissions to filter widgets	Pass
8	Open Dashboard | Editor	1. Open Web App 2. Login 3. Click on Dashboard name	Redirect to dashboard page where the user will have all the permissions to create/edit/filter widgets	Pass
9	Share Dashboard | Editor Mode	1. Open Web App 2. Login 3. Click on Dashboard name 4. Hover settings button 5. Click "share" button 6. Select "Editor" in the dropdown 7. Add user email 8. Click "Done" button	User can access and edit dashboard	Pass
10	Share Dashboard | Viewer Mode	1. Open Web App 2. Login 3. Click on Dashboard name 4. Hover settings button 5. Click "share" button 6. Select "Viewer" in the dropdown 7. Add user email 8. Click "Done" button	User can access and only apply filters on the dashboard	Pass
11	Change shared user permission	1. Open Web App 2. Login 3. Click on Dashboard name 4. Hover settings button 5. Click "share" button 6. Select user dropdown 7. Change shared option	User can still access the dashboard but their privileges were increased or decreased	Pass

Test Case #	Name	Steps	Expected Result	Result
12	Change dashboard start date	1. Open Web App 2. Login 3. Click on Dashboard name 4. Select "Filter" tab 5. Click "Date" filter card 6. Select Start Date 7. Change to "12/12/2022 12:00"	All the widgets on the dashboard should start at that date and the displayed date should change to "12/12/2022 12:00"	Pass
13	Change dashboard end date	1. Open Web App 2. Login 3. Click on Dashboard name 4. Select "Filter" tab 5. Click "Date" filter card 6. Select "End Date" 7. Change to "22/05/2023 22:00"	All the widgets on the dashboard should end at that date and the displayed date should change to "22/05/2023 22:00"	Pass
14	Change dashboard recurrence data to hour	1. Open Web App 2. Login 3. Click on Dashboard name 4. Select "Filter" tab 5. Click "Recurrence" filter card 6. Click on "hour"	The widgets should display hourly data	Pass
15	Change dashboard recurrence data to day	1. Open Web App 2. Login 3. Click on Dashboard name 4. Select "Filter" tab 5. Click "Recurrence" filter card 6. Click on "day"	The widgets should display daily data	Pass
16	Change dashboard recurrence data to mins	1. Open Web App 2. Login 3. Click on Dashboard name 4. Select "Filter" tab 5. Click "Recurrence" filter card 6. Click on "15 mins"	The widgets should display data from every 15 mins	Pass
17	Add global filter to dashboard	1. Open Web App 2. Login 3. Click on Dashboard name 4. Select "Filter" tab 5. Select "Data" tab 6. Drag data field 7. Drop onto "Global Filter" area 8. Select filter values	The widgets only display data based on the filters applied	Pass

Table E.1: Dashboard Test Cases

Test Case #	Name	Steps	Expected Result	Result
1	Create Widget - Line	1. Open Web App 2. Login 3. Click on Dashboard name 4. Select "Visualization" Tab 5. Select line chart icon 6. Insert title 7. Select "Data" title 8. Drag "Average Call Duration" 9. Drop on "Y Axis" dropzone 10. Drag "Time" 11. Drop on "X Axis" dropzone 12. Select "bluescale" color 13. Click on "Done"	New widget with a line chart and blue color created on the dashboard	Pass
2	Create Widget - Vertical Bars	1. Open Web App 2. Login 3. Click on Dashboard name 4. Select "Visualization" Tab 5. Select vertical bar chart icon 6. Insert title 7. Select "Data" title 8. Drag "Average Call Duration" 9. Drop on "Y Axis" dropzone 10. Drag "Time" 11. Drop on "X Axis" dropzone 12. Select "bluescale" color 13. Click on "Done"	New widget with a vertical bars chart and blue color created on the dashboard	Pass

3	Create Widget - Horizontal Bars	1. Open Web App 2. Login 3. Click on Dashboard name 4. Select "Visualization" Tab 5. Select horizontal bar chart icon 6. Insert title 7. Select "Data" title 8. Drag "Average Call Duration" 9. Drop on "Y Axis" dropzone 10. Drag "Time" 11. Drop on "X Axis" dropzone 12. Select "bluescale" color 13. Click on "Done"	New widget with a horizontal bars chart and blue color created on the dashboard	Pass
4	Create Widget - Scatter Chart	1. Open Web App 2. Login 3. Click on Dashboard name 4. Select "Visualization" Tab 5. Select scatter chart icon 6. Insert title 7. Select "Data" title 8. Drag "Average Call Duration" 9. Drop on "Y Axis" dropzone 10. Drag "Time" 11. Drop on "X Axis" dropzone 12. Select "bluescale" color 13. Click on "Done"	New widget with a scatter chart and blue color created on the dashboard	Pass

5	Create Widget - Pie Chart	1. Open Web App 2. Login 3. Click on Dashboard name 4. Select "Visualization" Tab 5. Select pie chart icon 6. Insert title 7. Select "Data" title 8. Drag "Average Call Duration" 9. Drop on "Values" dropzone 10. Drag "Time" 11. Drop on "Details" dropzone 12. Select "bluescale" color 13. Click on "Done"	New widget with a pie chart and blue color range created on the dashboard	Pass
6	Create Widget - Stacked Chart	1. Open Web App 2. Login 3. Click on Dashboard name 4. Select "Visualization" Tab 5. Select Stacked Chart icon 6. Insert title 7. Select "Data" title 8. Drag "Average Call Duration" 9. Drop on "Y Axis" dropzone 10. Drag "Average Ring Time" 11. Drop on "Y Axis" dropzone 12. Drag "Time" 13. Drop on "X Axis" dropzone 14. Select "bluescale" color 15. Click on "Done"	New widget with a stacked chart and blue color created on the dashboard	Pass

7	Create Widget - Combo Chart	1. Open Web App 2. Login 3. Click on Dashboard name 4. Select "Visualization" Tab 5. Select Combo Chart icon 6. Insert title 7. Select "Data" title 8. Drag "Average Call Duration" 9. Drop on "Bar Y Axis" dropzone 10. Drag "Average Ring Time" 11. Drop on "Line Y Axis" dropzone 12. Drag "Time" 13. Drop on "X Axis" dropzone 14. Select "bluescale" color 15. Click on "Done"	New widget with a Combo chart and blue color range created on the dashboard	Pass
8	Create Widget - No Title	1. Open Web App 2. Login 3. Click on Dashboard name 4. Select "Visualization" Tab 5. Select scatter chart icon 6. Select "Data" title 7. Drag "Average Call Duration" 8. Drop on "Bar Y Axis" dropzone 9. Drag "Average Ring Time" 10. Drop on "Line Y Axis" dropzone 11. Drag "Time" 12. Drop on "X Axis" dropzone 13. Select "bluescale" color 14. Click on "Done"	Error warning user that title cannot be empty	Pass
9	Move Widget	1. Open Web App 2. Login 3. Click on Dashboard name 4. Select widget 5. Drag widget to different position	Widget maintains new position even on refresh	Pass

10	Resize Widget	1. Open Web App 2. Login 3. Click on Dashboard name 4. Select widget corner 5. Drag widget corner	Widget maintains new size even on refresh	Pass
11	Add filter to widget	1. Open Web App 2. Login 3. Click on Dashboard name 4. Select widget or click on "Filter" button 5. Select "Data" tab 6. Drag field "User ID" 7. Drop onto "Filters on Widget" dropzone 8. Click filter card 9. Select users	Widget should only show data from the users selected	Pass
12	Remove Filter from widget	1. Open Web App 2. Login 3. Click on Dashboard name 4. Select widget 5. Click on "User ID" filter card 6. Click on "Trash" icon	Filter removed from widget shows all the data	Pass
13	Edit Widget - Title	1. Open Web App 2. Login 3. Click on Dashboard name 4. Select widget or click on "Edit" button 5. Change title to "Update Widget" 6. Click "Update" button	Widget title changed to "Update Widget"	Pass
14	Edit Widget - Color Range	1. Open Web App 2. Login 3. Click on Dashboard name 4. Select widget or click on "Edit" button 5. Change to "redscale" color 6. Click "Update" button	Widget color range changed to redscale	Pass

15	Edit Widget - Axis	1. Open Web App 2. Login 3. Click on Dashboard name 4. Select widget or click on "Edit" button 5. Clicks on "x" on the fields on the axis 6. Select "Data" tab 7. Drag "Total Events" 8. Drop on "Y Axis" dropzone 9. Drag "Call Type" 10. Drop on "X Axis" dropzone 11. Click "Update" button	Widget on the same chart type but different data	Pass
16	Delete Widget	1. Open Web App 2. Login 3. Click on Dashboard name 4. Select widget 5. Click "Trash" button 6. Confirm deletion	Widget deleted from dashboard	Pass
17	Search Data	1. Open Web App 2. Login 3. Click on Dashboard name 4. Select "Data" tab 5. Search for "Average"	Display all the fields that have average in the name: "Average Ring Time", "Average Waiting Queue", "Average Call Duration"	Pass
18	Search Filter	1. Open Web App 2. Login 3. Click on Dashboard name 4. Select widget or click on "Filter" button 5. Select "Data" tab 6. Drag field "User ID" 7. Drop onto "Filters on Widget" dropzone 8. Click filter card 9. Select users	Display all the fields that have average in the name. "External"	Pass

Table E.2: Widget Test Cases