



**Politecnico
di Torino**

Politecnico di Torino

Computer Engineering

A.y. 2022/2023

Graduation session April 2023

CoolVision: Innovative Web Application for automated budgeting and staffing

Supervisor:

Luigi De Russis

Gaspare Pappalardo

Candidate:

Stefano Rainò

Table of Contents

List of Figures	IV
1 Introduction	1
1.1 Context	1
1.2 Final goal of the thesis	3
1.3 Work organization	4
2 Preliminary analysis	5
2.1 Company starting point	5
2.2 Analysis of the current solution	6
3 Requirements analysis and planning	8
3.1 Interviews with project managers	8
3.2 Drafting of functional requirements	9
3.3 Potential use case	10
3.4 Identification of development methodology	10
3.4.1 Semi-Agile approach	11
3.4.2 Version prioritization	12
3.4.3 Build and testing	14
4 Evaluation of technological alternatives	16
4.1 Database management	16
4.2 Jira integration	17
4.3 Front-end and Back-end libraries	18
5 Implementation	20
5.1 Project management	20
5.1.1 Creating a project	20
5.1.2 View the details of the projects	23
5.1.3 Search for a project	27
5.1.4 Testing phase	28

5.2	Management and assignment of the effort	30
5.2.1	Adding effort to a project	30
5.2.2	Monthly visualization of the effort	32
5.2.3	Testing phase	33
5.3	Staffing management of individual projects	33
5.3.1	Assigning resources to a project	33
5.3.2	Use of DataGrid for assigning working days	35
5.3.3	Testing phase	41
5.4	Worklogs and Staffing visualization	42
5.4.1	Visualization of the history of the worklogs	43
5.4.2	View staffing for a project in the current and previous/next month	44
5.4.3	Testing phase	48
5.5	Visualization of report statistics	49
5.5.1	Consultation of Worked vs Estimated monthly for each project	49
5.5.2	Testing phase	51
5.6	Login and security management with last updates	51
5.6.1	Login via Google	52
5.6.2	Update a project	54
5.6.3	Navbar navigation menu	55
5.6.4	Testing phase	56
6	Conclusions	57
6.1	End results and final interviews	57
6.2	Application usage analysis	58
6.3	Potential future changes	60
	Bibliography	62

List of Figures

1.1	Example of a Jira software board	2
1.2	Agile vs Waterfall approach regarding project management	3
2.1	Example of composition of the first sheet for employee and project monitoring populated with dummy data	6
2.2	Example of composition of the workforce allocation sheet on projects populated with dummy data	7
2.3	Example of composition of sheet associated with individual employee allocation statistics populated with dummy data	7
3.1	Status of the CoolVision releases board, during development, on the Coolshop Jira server	12
3.2	Implementation progress of version 3 of the application monitored via the Jira server	13
3.3	Example of the Gitlab pipeline used for the code pushed into the project repository	14
3.4	CoolVision docker image running on port 27017	14
3.5	Gitlab commits history with pipeline applied on merge request . . .	14
4.1	Example of a document structure in MongoDB	17
4.2	Schema related to how the integration with Jira server through REST APIs was performed	18
5.1	Screen with the list of all projects and graphics associated with the daily worklogs	23
5.2	Project details visualization	26
5.3	Components used related to the creation of a project. Steps and subsequent alerts in case of successful or unsuccessful check jira code	28
5.4	New screen relating to the list of projects with graphics associated with the monthly worklogs of the reference month	29
5.5	Form associated with the addition of effort for a given project . . .	30
5.6	Effort list visualization for a single project	32

5.7	Visualization screen of Commercial vs Estimated vs Assigned for projects in November 2022	34
5.8	Table related to the possible assignments for the CoolVision project for November 2022	37
5.9	Visualization screen of Commercial vs Estimated vs Assigned for projects with the new section for previous/next month	42
5.10	Graph showing the history of working hours paid for a specific project divided into categories	43
5.11	Visualization of assignments on individual projects in the reference months	44
5.12	Data stored on the MongoDB side regarding the CoolVision project and the sales referring to CoolVision	46
5.13	New project details view with piecharts for commercial and estimate statistics	48
5.14	Report of CnhApp project for November 2022	50
5.15	Report page with statistics regarding previous/next month	51
5.16	Login interface	54
5.17	Navbar and drawer in coolvision web app with logo of the application	56
6.1	Main features related to the v1.0.1 of Coolvision stored in Coolshop's Jira server	60

Chapter 1

Introduction

Project management is critical to the success of any organization. In particular, it is important that all those involved have a clear and complete vision of the projects that follow, as well as an accurate understanding of deadlines and costs. In order to improve this aspect and obtain greater efficiency, Coolshop S.r.l decided to develop an internal web application dedicated to project management

1.1 Context

Corporate project management refers to the process of managing projects within a company, including planning, organizing, and executing projects to meet specific goals and objectives. Effective corporate project management involves the use of project management tools and techniques to ensure that projects are completed on time, within budget, and to the desired level of quality. There are several applications available for managing corporate project management, such as **Microsoft Project** [1], **Asana** [2], **Trello** [3], and **Jira** [4]. These tools allow project managers to schedule tasks, allocate resources, track progress, and collaborate with team members.

Effectively managing business projects requires ensuring that they are aligned with business objectives and that resources are allocated appropriately. This involves identifying project stakeholders, developing a comprehensive project plan, and establishing a clear communication plan to ensure effective collaboration with all parties involved. Additionally, project managers must regularly monitor and evaluate project progress and manage project risks to ensure successful project outcomes. To achieve these goals, it is important for project managers to regularly review and update project schedules and budgets, making necessary adjustments to keep them on track. Following best practices in corporate project management can help ensure that projects are executed efficiently and effectively.

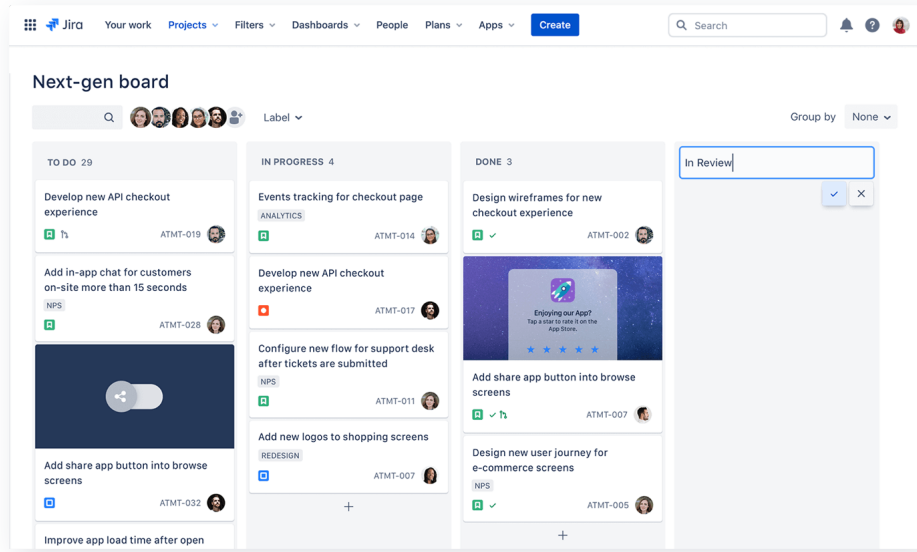


Figure 1.1: Example of a Jira software board

Currently, many organizations still use traditional project management approaches, such as the **Waterfall** [5] methodology. However, there is a growing trend towards more agile project management methodologies, such as **Scrum** [6] and **Kanban** [7], which prioritize flexibility, collaboration, and iterative development. In particular:

- **Waterfall** is a linear, sequential approach to project management. It involves completing one phase of the project before moving on to the next, with no room for iteration or changes. This approach is best suited for projects that have well-defined requirements and are not likely to change during the project lifecycle. Waterfall methodologies are known for their predictability, as each phase has clearly defined deliverables and timelines.
- **Scrum and Kanban** are iterative and incremental *Agile methodologies* [8]. They are best suited for projects where requirements are likely to change or evolve during the project lifecycle. Agile methodologies involve breaking the project down into smaller, more manageable pieces and working on them in short iterations or sprints. This approach allows for more flexibility and adaptability throughout the project lifecycle, as changes and feedback can be incorporated into the project as it progresses.

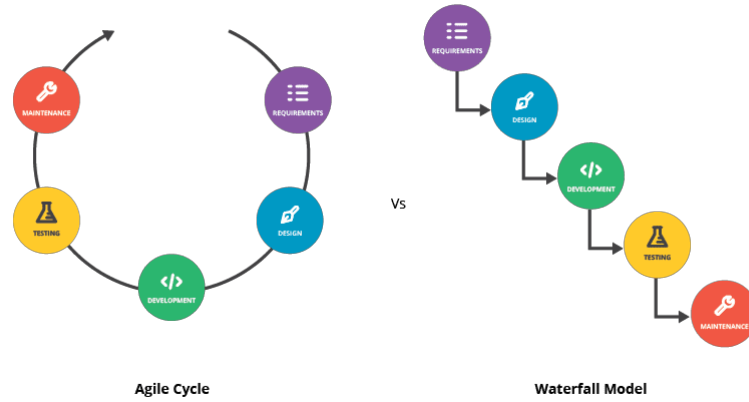


Figure 1.2: Agile vs Waterfall approach regarding project management

1.2 Final goal of the thesis

The new application, which will be described in detail in the following sections, has been developed with the aim of providing an intuitive and effective solution for managing corporate projects. In particular, it will be integrated with Jira software to provide real-time information on worklogs, deadlines and costs. The use of this new application will reduce the workload required to manage budgets, and will also help team leaders to keep development costs and deadlines under control compared to initial estimates.

In this thesis, the functionalities of the new application will be described in detail, as well as the development and testing processes that led to its implementation. Furthermore, analysis will be provided on the performance and impact of the new application on project management within Coolshop S.r.l. The goal is to provide all stakeholders with a comprehensive view of the projects they are involved in, so that they can have a clear understanding of their progress. By creating this web application, the company expects to increase efficiency and productivity in the management of projects, as well as improve communication between all stakeholders involved. Additionally, it will provide an accurate picture of project budgets and staffing, enabling more informed decision-making.

In conclusion, the final goal of this thesis is to develop a web application that streamlines project management, simplifies budget and staffing management, and provides stakeholders with the information they need to make informed decisions. The aim is to create a solution that helps the company to reach its goals more effectively and efficiently, with the ultimate goal of driving business success.

1.3 Work organization

In this section, it will be discussed the expected work organization for the development of the web application called **CoolVision**. The focus will be on the analysis of requirements and the development of the application itself. The main points covered will be:

- Chapter 2, **Preliminary Analysis**: The starting point for the application development is the current manual process of budget management using excel sheets. The objective of this project is to streamline this process and provide a centralized platform for project management, budget tracking, and staffing.
- Chapter 3, **Requirements**: The analysis of requirements will involve gathering information and feedback from various stakeholders within the company. This information will be used to define the functional and non-functional requirements of the application. Based on these requirements, a functional specification plan will be created, outlining the expected functionality of the application.
- Chapter 5, **Development**: The development of the application will involve the use of React [9] for the front-end and Node.js [10] for the back-end. The development will follow a Continuous Integration/Continuous Deployment (CI/CD) process, using GitLab [11] for version control.
- Chapter 5 and 6, **Testing**: The application will be developed in iterative sprints, with regular check-ins with stakeholders to ensure that the application meets their needs and requirements. The application will be tested thoroughly before deployment, to ensure that it meets the quality standards set out in the functional specification plan.

In conclusion, the work organization for the development of this web application will be structured and focused on delivering a high-quality product that meets the needs of the project managers at Coolshop. The process will involve a thorough analysis of requirements, iterative development, and regular check-ins to ensure that the final product meets the expectations.

Chapter 2

Preliminary analysis

The preliminary analysis phase is essential to then identify and outline the key requirements and objectives of the project. This phase sets the foundation for the entire project and helps to ensure that the project aligns with the expectations. This chapter discusses the key components of the preliminary analysis phase and provides insight into how was approached this critical stage of the project.

2.1 Company starting point

Prior to the implementation of a web application for project staffing management, Coolshop relied on the **use of spreadsheets** to manage its projects. The company's starting point was characterized by a manual and fragmented approach to project staffing. The process started with project managers manually creating spreadsheets to track the allocation of resources to each project. These spreadsheets would then be shared with the relevant stakeholders for review and approval. Once approved, the spreadsheets would be used to track the progress of the project, including the number of hours worked by each team member and the overall budget. Despite being a common method for project management, using spreadsheets to manage project staffing had several disadvantages.

Firstly, the manual process was time-consuming and prone to errors. It was also challenging to access real-time data, making it difficult for managers to make informed decisions. In addition, the use of spreadsheets lacked the transparency and accountability required to ensure effective collaboration between team members. Furthermore, the lack of a centralized platform made it challenging to track the progress of multiple projects simultaneously. This resulted in duplication of effort and a significant amount of time spent on manual data entry. As the company grew, it became increasingly difficult to manage the increasing number of projects and resources using this manual process. In conclusion, the use of spreadsheets

to manage project staffing is a suitable starting point for Coolshop, but it is clear that a more efficient and effective solution is required.

2.2 Analysis of the current solution

The current solution is composed of several sheets, each with a specific purpose.

The first sheet is dedicated to monitoring employees and projects. It lists all the projects and employees, divided by their department of belonging. The department of belonging represents an internal division that refers to **developers (Heroes)**, **UX/UI designers (Jedis)**, and **functional roles (Paladins)**. This sheet provides an overview of the allocation of employees between different projects, and it is also taken as a starting point for the composition of the following ones since the presence of projects, employees and different departments in the following sheets refer to the lists present here. An example of sheet composition can be seen in 2.1

Ruolo	COUNT		Cognome	Nome	Ruolo new		Progetto
Dev (backender)	1		DEV1	DEV1	Dev (backender)	▼	-
Dev (frontender)	1		DEV2	DEV2	Dev (frontender)	▼	Ferie
Dev (full stack)	1		DEV3	DEV3	Dev (full stack)	▼	Progetto1
Hero	1		FUNZ1	FUNZ1	Paladin J	▼	Progetto2
Jedi	1		JEDI1	JEDI1	Jedi	▼	Progetto3
Paladin	1		HERO1	HERO1	Hero	▼	Formazione
Paladin J	1		FUNZ2	FUNZ2	Paladin	▼	
DEV	4	66.67%					
FUNZ	2	33.33%					

Figure 2.1: Example of composition of the first sheet for employee and project monitoring populated with dummy data

The second sheet is dedicated to the allocation of individual employees. Each row represents a **pair of (employee, project)**, while the various columns represent the months of the year. The columns are divided into sub-columns: **To Check, Allocation, To Allocate, Days, and Notes**. In the *Days* column, it is possible to enter the number of days to allocate for that employee in that project for that month. Automatically, the *Allocation* column will be populated with the percentage of allocation made relative to that project in that month on the total working days available to the employee. The *To Allocate* column will also be automatically populated with the remaining percentage of days to allocate to that person in that month relative to the total number of days in the month (working days). An example of the composition of this second sheet can be seen in 2.2

The third sheet keeps track of the total **working days and holidays** of employees belonging to specific departments for each project and month. This sheet

				Jan 2022				Feb 2022			
Working Month Days				19				20			
Cognome	Ruolo	Progetto		Allocazione	Da Allocare	Giorni	Note	To Check	Allocazione	Da Allocare	Giorni
DEV1	Dev (backender)	Progetto1		39%	0%	7.5			50%	0%	10
DEV1	Dev (backender)	Progetto2		50%	0%	9.5			50%	0%	10
DEV1	Dev (backender)	Ferie		11%	0%	2			0%	0%	0
DEV1	Dev (backender)	Formazione		0%	0%	0			0%	0%	0
DEV2	Dev (frontender)	Progetto3		74%	0%	14.15			100%	0%	20
DEV2	Dev (frontender)	Ferie		11%	0%	2			0%	0%	0
DEV2	Dev (frontender)	Formazione		15%	0%	2.85			0%	0%	0
DEV3	Dev (full stack)	Progetto2		70%	0%	13.3			30%	0%	6
DEV3	Dev (full stack)	Progetto3		20%	0%	3.8			70%	0%	14
DEV3	Dev (full stack)	Ferie		10%	0%	1.9			0%	0%	0
DEV3	Dev (full stack)	Formazione		0%	0%	0			0%	0%	0
JEDI1	Jedi	Progetto1		37%	0%	7			30%	30%	6
JEDI1	Jedi	Progetto2		21%	0%	4			20%	30%	4
JEDI1	Jedi	Progetto3		32%	0%	6			20%	30%	4
JEDI1	Jedi	Ferie		11%	0%	2			0%	30%	0
JEDI1	Jedi	Formazione		0%	0%	0			0%	30%	0
FUNZ1	Paladin J	Progetto1		89%	0%	17			98%	0%	19.6

Figure 2.2: Example of composition of the workforce allocation sheet on projects populated with dummy data

provides an overview of the workload for each department and how much time is allocated to each project. While, the fourth sheet keeps track of the percentage of days allocated to each employee for each month.

ALLOCAZIONE PERSONE					
Cognome	Dipartimento	Jan 2022	Feb 2022	Mar 2022	Apr 2022
DEV1	DEV1	100%	100%	100%	100%
DEV2	DEV2	100%	100%	100%	100%
DEV3	DEV3	100%	100%	100%	100%
FUNZ1	FUNZ1	100%	100%	100%	100%
FUNZ2	FUNZ2	100%	100%	100%	100%
HERO1	HERO1	100%	105%	100%	100%
JEDI1	JEDI1	100%	70%	80%	100%

Figure 2.3: Example of composition of sheet associated with individual employee allocation statistics populated with dummy data

One of the main disadvantages of this solution is that it is difficult to manage allocations on different projects and months for each employee. For example, if an employee needs to work on two different projects in the same month, it is challenging to allocate the correct number of days to each project while ensuring that the total number of working days is not exceeded. Moreover, since the solution is based on an Excel file, there is a risk of errors due to manual data entry. For example, if the number of working days for an employee is not entered correctly, this can lead to incorrect allocations and inaccurate workload calculations. Furthermore, it is also difficult to manage the vacation periods of individual employees in the reference months. In fact, as can be seen from the sheets, holidays are considered as if they were a full-fledged project, creating confusion in allocation and monitoring. In the next chapter, will be described the implementation of CoolVision solution and how it addresses these issues.

Chapter 3

Requirements analysis and planning

The requirements analysis involves identifying, analyzing, documenting, and prioritizing the requirements of a software project. This chapter will discuss the techniques used in the requirements analysis process. Additionally, the chapter will highlight the importance of project planning and how it helps in allocating resources and managing project timelines. Overall, it aims to provide a comprehensive understanding of the requirements analysis and planning process, and how it sets the foundation for the success of the development.

3.1 Interviews with project managers

In order to better understand the needs and wants of the project managers within Coolshop, a series of interviews were conducted. The goal of these interviews was to determine what the project managers are looking to improve in their current budgeting and staffing processes, and what they would like to see in a web application that could assist them. The following are some of the key insights that were gained from these interviews:

- **Streamlined Processes:** Many of the project managers noted that the current budgeting and staffing process is quite manual, and takes a significant amount of time to complete. They expressed a desire for a more streamlined process, where data can be easily input and accessed, and calculations can be done automatically.
- **Improved Collaboration:** Many of the project managers stressed the importance of better collaboration and communication between all stakeholders involved in the project. They expressed a desire for a web application that

could provide real-time access to information and allow for easy collaboration between team members.

- **Accurate Reporting:** The project managers emphasized the importance of accurate and up-to-date reporting in order to effectively manage the project. They stated that they would like a web application that can provide accurate and comprehensive reporting, making it easier to track progress and make informed decisions.
- **User-Friendliness:** The project managers expressed a desire for a web application that is user-friendly and easy to navigate, with a simple and intuitive interface. They stated that this would help to minimize the learning curve, making it easier for all stakeholders to get up to speed and start using the application quickly.

These insights provide valuable information about what the project managers within Coolshop are looking for in a web application to assist with budgeting and staffing. By understanding their needs, it is possible to create an application that meets their requirements and helps them to manage their projects more effectively.

3.2 Drafting of functional requirements

Based on the insights gathered from the interviews with the project managers, it was possible to establish a list of functional requirements, to then be translated into implementation features, which can be considered focal points for the creation of the web app. They can be summarized in:

1. **Budgeting and Cost Tracking:** The application should provide the ability to track and manage project budgets, including the ability to allocate resources and track expenses. This will help to ensure that the project stays within budget.
2. **Resource Allocation:** The application should provide the ability to easily allocate resources to each project, including both personnel and equipment. This will help to ensure that the team is able to work effectively and efficiently, and that the project stays on track.
3. **Project Scheduling:** The application should provide the ability to schedule and manage project timelines, making it easier to track progress and make informed decisions about how to allocate resources.
4. **Performance Tracking:** The application should provide the ability to track the performance of each team member, allowing for continuous improvement and the ability to identify areas for improvement.

These are the main features that emerged from the analysis carried out and represent the starting points on which the programming of the application development was built. Thanks to them it was possible, as will be seen in the following chapters, to prioritize the development and focus the work on these requirements.

3.3 Potential use case

Once the main features that an application of this type must be able to support have been identified, a potential use case has been drawn up that can be used as a reference during development. From the point of view of a **project manager**:

- He/She logs into the application through his corporate account and creates a project within the web application.
- He/She can decide to indicate the estimated and commercial days for the realization of each project.
- He/She can view the list of projects created and present within the app and for each of them he/she can see the details relating to:
 1. List of allocations made for each single period.
 2. List of estimated resources for each single period.
 3. List of associated worklogs for each single period.
- He/She can allocate resources to individual projects on the basis of their availability in the reference periods.

Considering this use case it has been possible to develop a working methodology that can meet the requirements and allow to recreate this use case. The following sections will explain how.

3.4 Identification of development methodology

The development methodology for the CoolVision web application was a critical decision that needed careful consideration. After reviewing the project requirements, it was clear that an Agile approach would be the best fit for the project. However, given the project constraints and the need to balance flexibility with predictability, an **semi-Agile** approach was chosen. One of the main reasons for choosing a semi-Agile approach was to accommodate the needs of the project managers while ensuring the project's success. This approach provided a structured framework for the development while also allowing for flexibility and adaptation to changing requirements. This balance was achieved through the implementation of a

modified Agile methodology, which would enable to remain focused on the project's objectives while remaining responsive to changing needs. Another important factor in the decision-making process was the need to **deliver the project within the allocated budget and timeline**. A fully Agile approach can be challenging to manage, particularly when it comes to delivering projects within a fixed budget and timeline. A semi-Agile approach, on the other hand, provides the team with the necessary structure and guidelines to manage costs and timelines effectively while remaining flexible enough to incorporate feedback and changes as needed.

3.4.1 Semi-Agile approach

Considering the semi-Agile approach, the development of the application was divided into **six different versions**, each associated with a specific sprint. Each of these versions includes different features of the application that are prioritized in the next section. The development process requires that each version/sprint be developed over a period of one week, with a particular breakdown of **3 working days for development** and **2 working days for testing** and fixing. This approach was chosen for several reasons. Which are:

1. It allowed for the rapid development of the application in a short time frame. The semi-Agile approach also provided flexibility, as each sprint was focused on specific features and could be adjusted as needed based on feedback and changing requirements.
2. Breaking down the development into smaller, manageable chunks allowed for better tracking of progress and easier identification of potential issues.
3. The approach allowed for greater collaboration between the development team and other stakeholders, including the project managers and end-users, who were able to provide feedback on the functionality of each sprint.

Version	Status	Progress	Start date	Release date	Description
6	UNRELEASED	<div></div>			
5	UNRELEASED	<div></div>			
4	UNRELEASED	<div></div>			
3	UNRELEASED	<div></div>	03/Oct/22	07/Oct/22	
2	UNRELEASED	<div></div>	26/Sep/22	30/Sep/22	
1	UNRELEASED	<div></div>	19/Sep/22	23/Sep/22	

Figure 3.1: Status of the CoolVision releases board, during development, on the Coolshop Jira server

3.4.2 Version prioritization

The features that have been identified for the development of CoolVision have emerged from the chapter 3.2, where the needs of the project managers were analyzed. Based on their feedback and requirements, a list of features was compiled and prioritized to develop the application in a way that meets their needs. The version prioritization and testing plan is as follows:

- **Version 1** (Sprint 1):
 1. Creating a project
 2. View the details of a project
 3. Search for a project by name/jira code
- **Version 2** (Sprint 2):
 1. Adding effort to a project
 2. Monthly visualization of the effort of each project
- **Version 3** (Sprint 3):
 1. Assigning resources to a project
 2. Display of the list of employees with remaining days for each single month
- **Version 4** (Sprint 4):
 1. Visualization of the history of the worklogs within the details of each individual project

2. View staffing for a project in the current/previous/next month

- **Version 5** (Sprint 5):

1. Consultation of Worked vs Estimated man days monthly for each project

- **Version 6** (Sprint 6):

1. Google login authentication

For each version, as said before, it will be worked on the prioritized features for three days and then dedicate two days to testing and fixing any issues discovered during testing. The project manager will be responsible for overseeing the testing phase and making sure that all features are functioning as expected before the release of the new version. To ensure the quality of each release, the project manager will create test cases and checklists for each feature, and assign specific team members to perform the tests. Any bugs or issues discovered during testing will be logged and prioritized for fixing in future sprints or in the same. By following this process, the project manager can ensure that each version of CoolVision is fully functional and meets the requirements of the users, while also allowing for flexibility in the development process to address any unforeseen issues that may arise.

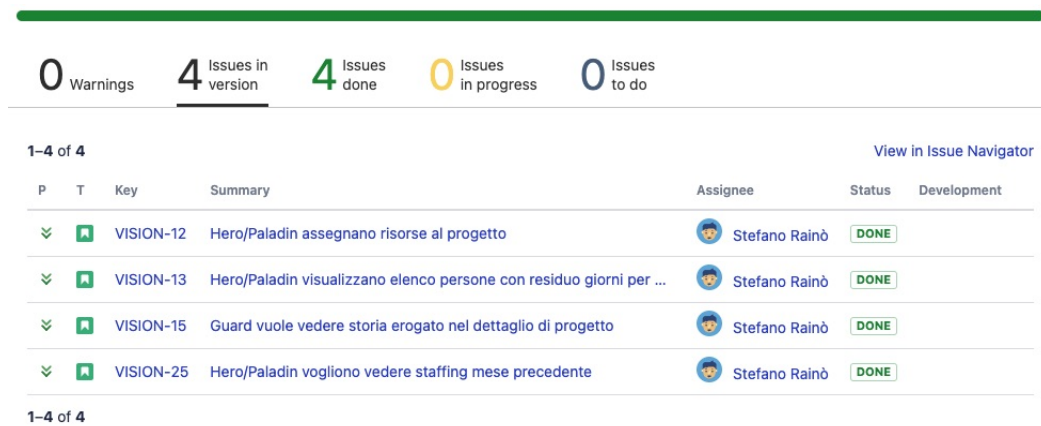


Figure 3.2: Implementation progress of version 3 of the application monitored via the Jira server

3.4.3 Build and testing

The complete build of the CoolVision application runs on the Coolshop company servers. This approach allows for a more secure and reliable deployment of the application, ensuring that it meets the company's standards and requirements. Additionally, the use of the company servers ensures that the application is easily accessible by all team members involved in the development process, regardless of their physical location. To ensure a smooth and efficient deployment process, the project team decided to use **GitLab pipelines** 3.3. A GitLab pipeline is a set of automated processes that allow for the building, testing, and deployment of code changes. The pipeline is triggered by the push of new code changes to the Git repository, and it follows a predefined set of steps to build the application and ensure that it is working correctly.

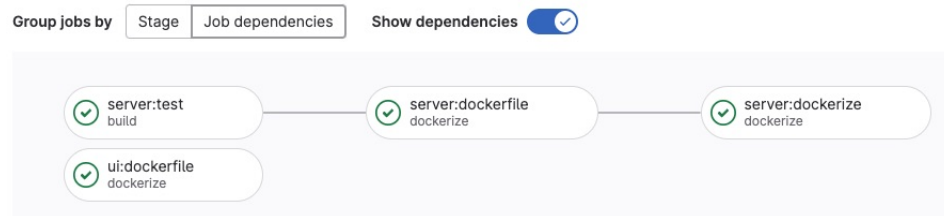


Figure 3.3: Example of the Gitlab pipeline used for the code pushed into the project repository

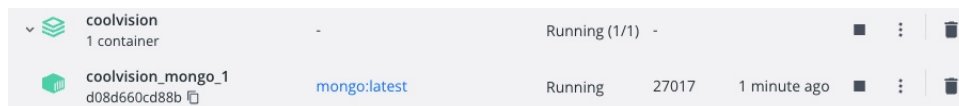


Figure 3.4: CoolVision docker image running on port 27017

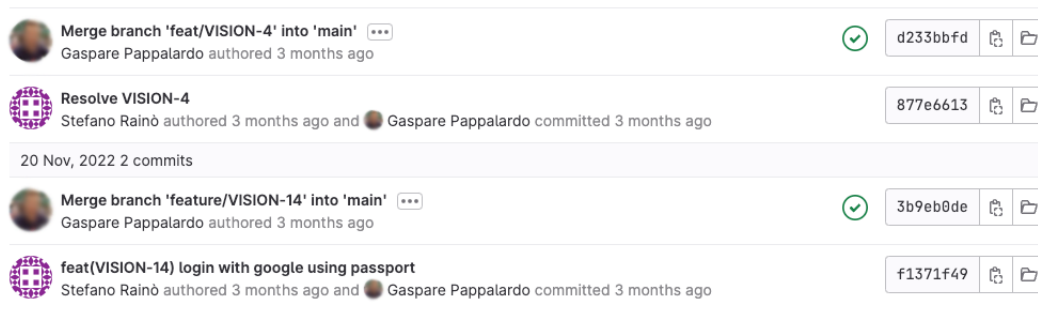


Figure 3.5: Gitlab commits history with pipeline applied on merge request

The use of a GitLab pipeline allows the team to automate much of the testing and deployment process, reducing the likelihood of errors and inconsistencies during the deployment phase. Additionally, it provides a clear and structured process for testing and deploying the application, ensuring that all team members are following the same procedures and guidelines. The application build process was supported by the use of **Docker** [12]. Docker is a powerful tool for creating, managing, and deploying containerized applications. Docker containers provide an isolated environment that encapsulates all the dependencies and libraries required to run an application, allowing it to be run on any system that supports Docker, regardless of the underlying hardware and software configuration.

Using Docker for CoolVision provides several benefits, including simplified application deployment and portability, improved resource utilization, and streamlined development and testing processes. By packaging the application and all its dependencies into a single Docker image, the team can easily deploy and manage the application across multiple environments, including local development/build and testing machines, staging servers, and production systems. Furthermore, the integration of Docker with GitLab pipelines allows to automate much of the testing and deployment process. The Docker image can be built and tested within the pipeline, ensuring that it is consistent and error-free before being deployed to the target environment. This automated approach reduces the likelihood of errors and inconsistencies during the deployment phase and provides a clear and structured process for testing and deploying the application.

Chapter 4

Evaluation of technological alternatives

The process used for selecting the technological alternatives involved researching and evaluating several technologies that could be used for both the front-end and back-end of the application. It was reviewed several factors such as **functionality**, **scalability**, **community support**, and **ease of development and maintenance**. After careful evaluation, it was decided to use **React** for the front-end and **Node.js** for the back-end. They were chosen because:

- **React**: It provides flexibility, high-performance and extensive documentation. It also has a large community of developers and a vast number of libraries and tools that can be used to create complex user interfaces.
- **Node.js**: It provides scalability, efficiency, and ability to handle a high volume of requests. It also provides a vast number of libraries and tools, making it easier to develop and maintain the application.

4.1 Database management

For the database management of CoolVision, the team decided to use **MongoDB** [13] as the database and **GraphQL** [14] as the query language. There were several factors that led to this decision.

Firstly, MongoDB is a **NoSQL document-oriented database**, which means it stores data in JSON-like documents instead of in tables with rows and columns like traditional SQL databases. This makes it more flexible and scalable, especially for applications with rapidly changing data structures. Additionally, MongoDB's ability to handle large amounts of data and its built-in replication and sharding capabilities make it a popular choice for modern web applications.

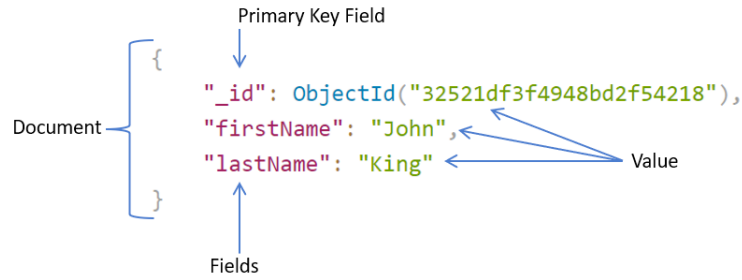


Figure 4.1: Example of a document structure in MongoDB

Secondly, GraphQL was chosen as the query language for its ability to efficiently retrieve data from the database. GraphQL allows developers to specify exactly what data they need from the database, reducing the amount of data returned and increasing the performance of the application. This is particularly useful for mobile applications or other low-bandwidth environments where data transfer speed is critical.

Furthermore, GraphQL's type system provides a clear and structured way of defining the data model, making it easier for developers to understand and maintain the application codebase. Additionally, GraphQL's ability to join data from multiple sources and its support for real-time subscriptions make it a powerful tool for building interactive and dynamic web applications. Overall, the combination of MongoDB and GraphQL provides a flexible, scalable, and performant solution for managing the data of CoolVision.

4.2 Jira integration

Integrating Coolvision with Coolshop's Jira server was a crucial requirement for the success of the project. The Jira server is the primary tool used by Coolshop's employees for tracking their work and progress on various projects. Therefore, having the ability to access and manipulate this data from within Coolvision was essential for the application's success. To achieve this integration, it was explored various options, including the use of **Jira's REST API** [15] and webhooks. Ultimately, it was decided to use the REST API to fetch data from the Jira server, as it provided more flexibility and control over the data retrieval process.

There were identified two key pieces of information that were needed to be retrieved from Jira: **project data and worklogs**. Project data includes information such as project name, project description, project lead, and start and end dates. Worklogs, on the other hand, contain information about the time spent by each developer on a particular project.

To retrieve this data, the Jira REST API were used and integrated it into the GraphQL schema used by the application. Custom resolvers have been written that would fetch the necessary data from the Jira server, and then transformed and returned it to the GraphQL client. The APIs are responsible for retrieving data related to projects and worklogs from the Jira Server instance and transforming them into the format expected by Coolvision. The integration will be implemented using Node.js and the **Express.js** [16] framework.

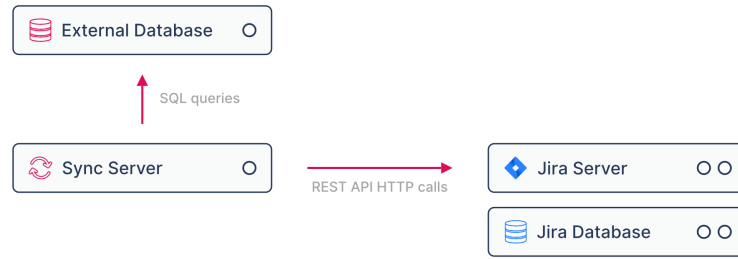


Figure 4.2: Schema related to how the integration with Jira server through REST APIs was performed

4.3 Front-end and Back-end libraries

The first approach taken with the development of the application was associated with the search for possible libraries that could have been useful for the implementation of the features associated with the individual versions. In particular, the following main libraries were identified (and subsequently used):

- **Front-end libraries:**

1. **Recharts** [17]: It is a charting library for React that makes it easy to create beautiful and responsive charts for data visualization. It was chosen because it is highly customizable and has a wide range of chart types to choose from, which allowed for the creation of the monthly effort visualization feature in CoolVision.
2. **Material UI** [18]: It is a React-based UI library that provides a set of ready-to-use components and styles based on Google's Material Design guidelines. It was chosen for its ease of use, high-quality components, and the fact that it integrates well with React. Material UI was used extensively throughout CoolVision for building the user interface.

- **Back-end libraries:**

1. **Mongoose** [19]: It is a MongoDB object modeling library for Node.js that provides a higher-level API for interacting with MongoDB databases. It was chosen for its ease of use, flexibility, and robust features, including validation and middleware. Mongoose was used extensively throughout Coolvision for managing database interactions.
2. **Express-GraphQL** [20]: It is a GraphQL server library for Node.js that allows for building GraphQL APIs with ease. It was chosen for the fact that it integrates well with other Node.js libraries like Mongoose. Express-GraphQL was used for building the GraphQL API for Coolvision.
3. **Jira.js** [21]: It is a JavaScript library that provides a simple and consistent interface for interacting with the Jira REST API. It was chosen for its robust features, including support for authentication and error handling. Jira.js was used for integrating CoolVision with the Jira server at Coolshop, allowing for the retrieval of project and worklog data.

Chapter 5

Implementation

The objective of this chapter is to analyze in detail the implementation phase of the project. Here it will be described the development process of the application starting from version 1, tracking the progress of the features that were developed. It will be explained the challenges encountered during the development phase and how they will be solved to achieve the final product. Additionally, it will provide a detailed overview of the various technologies and tools utilized during the development process.

5.1 Project management

This section is dedicated to the development of the first version of the application. It is related to the implementation of the features: creation of a project, displaying the details of a project and searching for it. The methods and technologies used to develop these features will be analyzed, as well as an analysis and examination linked to their testing phase.

5.1.1 Creating a project

As mentioned before, the main feature of this version is represented by the creation of a project. For this first feature it was decided to use the **Modal component**, imported by default from mui/material library. It provides the insertion of two fields, namely: **Project name** and **Jira code**. The first field refers to the name for which the project will be stored within the database on the MongoDB side, while the second represents the code associated with the project stored in the corporate Jira server. This last field will be of fundamental importance as regards the developments of subsequent features, because thanks to it and to the Jira REST API, it will be able to monitor the days worked by the employees. The code for

adding the project to the database is as follows:

```

1 async function addProject(project: Project) {
2   return new Promise((res, rej) => {
3     graphqlQuery(addProjectQuery(project.name, project.jira))
4       .then(_ => res("Project inserted successfully ..."))
5       .catch(err => rej(err));
6   });
7 }

```

Listing 5.1: addProject function in ./client/src/API.ts

```

1 const addProjectQuery = (name: String, jira: String) => {
2   return ('mutation {
3     addProject (name: "${name}", jira: "${jira}") {
4       _id,
5       name,
6       jira,
7       sold {month, assignments {_id_user, assigned}}
8     }
9   }'
10 )}

```

Listing 5.2: addProjectQuery function in ./client/src/Queries.ts

As can be seen from the Query, it refers to a **mutation**. A mutation in GraphQL represents a convention used to establish that the operation being performed will cause a write to the reference database. In particular, it is translated into code thanks to the use of the GraphQL library, which prepares the mutation, and thanks to the **resolver** that translates this mutation into asynchronous operations to be performed by the server. The name of the mutation in this case is **addProject**, and it takes the project name and its Jira code as parameters. It returns the data associated with the project just entered, also with reference to the "sold" for that project, which in this case is empty. The following snippets instead explain the flow of adding the new project:

```

1 addProject: {
2   type: Project,
3   args: {
4     name: {type: GraphQLString},
5     jira: {type: GraphQLString}
6   },
7   resolve: async (_, args) => {
8     return addProject(args as AddProjectProps)
9   }

```

Listing 5.3: addProject mutation in ./server/graphql/mutation.ts

```
1 const addProject = async function ({name, jira}: AddProjectProps){
2   const project = new ProjectSchema({
3     name: name,
4     jira: jira,
5     sold: []
6   });
7   return await project.save();
8 }
```

Listing 5.4: addProject resolver in ./server/resolvers/resolver.ts

Once the mutation has been defined, it is possible to indicate how it should be resolved through the corresponding resolver. The resolver refers to the schema called **ProjectSchema**, which is a new schema specially created by Mongoose that describes the characteristics of the new object which is going to be saved. For every API that makes a MongoDB call, like this one, all these functions have been implemented for the build and proper handling of stored and returned individual data.

```
1 const projectSchema = new Schema({
2   name: {
3     type: String,
4     required: true,
5   },
6   jira: {
7     type: String,
8     required: true,
9   },
10  sold: [
11    {
12      month: { type: String, required: true },
13      assignments: [
14        {
15          _id_user: { type: String, required: true },
16          assigned: { type: Number, required: true }
17        }
18      ]
19    }
20  ]
21 });
```

Listing 5.5: projectSchema mongoose schema in ./server/schemas/project.ts

5.1.2 View the details of the projects

The next step was to create the screen for displaying the details of a project. To do this it started from the screen displaying **all the projects stored in the database**. The figure 5.1 shows that it was decided to implement this screen by dividing it into two parts:

- The first concerning the daily **statistics related to the worklogs** recorded on Jira, up to that moment
- The second relating to the **list of projects** stored in the database

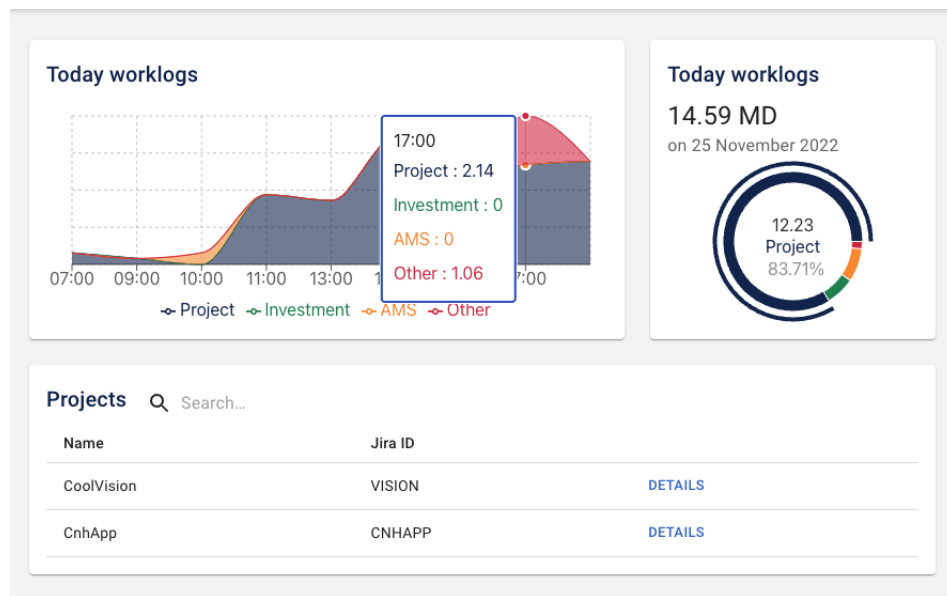


Figure 5.1: Screen with the list of all projects and graphics associated with the daily worklogs

This type of visualization was chosen for two reasons: the first to have an immediate view of the percentage and quantity of working hours recorded by the employees (on the relevant day of reference) and the second to display one more information that would otherwise have been lost. For the creation of the graphs, the **AreChart** and **PieChart** components of the **Recharts** library were chosen. The first allows to have a quantitative view of the reference data, split into different areas associated with the type of project with respect to the worklogs refer, and related to the different hours for which these worklogs were registered. The second because allows to have a global view of how these worklogs are divided into individual project types. To create both graphs, the server-side **getJiraWorklogs**

API was used, which takes two dates as parameters, and returns all the worklogs in the form of an object containing:

- The hour in which this worklog was registered
- The worked hours in the form of the so called **Man Days** (a man day represent 8 worked hours)
- The type of the project with respect the worklog is referred

To create this type of API, the **Version2Client** object associated with the Jira.js library was used, which first of all allows to establish a connection with the reference Jira server, and secondly provides a series of methods for creating search queries associated with any aspect of the referring server. The following snippets show the realization procedure of what has been explained:

```

1  const client = new Version2Client({
2    host: env.external.jira.host,
3    authentication: {
4      oauth: {
5        consumerKey: env.external.jira.oauth.consumerKey,
6        consumerSecret:
7          "-----BEGIN PRIVATE KEY-----\n" +
8            `${env.external.jira.oauth.consumerSecret}\n` +
9            "-----END PRIVATE KEY-----",
10       accessToken: env.external.jira.oauth.accessToken,
11       tokenSecret: env.external.jira.oauth.tokenSecret
12     },
13   },
14 });

```

Listing 5.6: client created with Version2Client in ./server/src/dao.ts

To establish a connection with the server it is necessary to set various security parameters, which have been stored in a **private .env** file. These parameters were generated on the Jira server side using an account with reference server administrator privileges.

```

1  async function getJiraWorklogs(startDate: string, endDate: string)
2  {
3    return new Promise(async (res, _) => {
4      let worklogs: DailyWorklog[] = []
5      let start = 0
6      let newWorklogs = await client.issueSearch
7        .searchForIssuesUsingJql({
8

```

```

9      jql:
10        'worklogDate >= "${startDate}"
11        AND worklogDate <= "${endDate}" '
12      ,
13      fields: ['components', 'worklog'],
14      startAt: start,
15      maxResults: maxResults
16    })
17    while (start < Number(newWorklogs.total)) {
18      newWorklogs.issues?.filter(issue => {
19        return issue.fields.worklog.worklogs.length > 0
20      }).forEach(issue => {
21        issue.fields.worklog.worklogs
22          .forEach(worklog => {
23            if (dayjs(worklog.started) <= dayjs(endDate)
24              && dayjs(worklog.started) >= dayjs(startDate)) {
25              worklogs.push({
26                hour:
27                  dayjs(worklog.created).format('HH:00'),
28                worked:
29                  Number(worklog.timeSpentSeconds)/manDay
30              ,
31                type:
32                  issue.fields.components.length > 0 ?
33                  issue.fields.components.pop()?.name
34                  : 'Project'
35              })
36            }
37          })
38      })
39    })
40    start += maxResults
41    newWorklogs = await client.issueSearch
42      .searchForIssuesUsingJql({
43      jql: '
44        worklogDate >= "${startDate}"
45        AND worklogDate <= "${endDate}" '
46      ,
47      fields: ['components', 'worklog'],
48      startAt: start,
49      maxResults: maxResults
50    })
51  }
52  res(worklogs)
53 })
54 }

```

Listing 5.7: getJiraWorklogs in ./server/src/dao.ts

After doing this, the list of projects was instead managed through the use of the **Table** component of the mui library, which presents in the third column the rendering of a button that refers to the project detail page, which can be viewed in figure 5.2.

The figure displays two versions of a project details form. The top version is a filled-out card for a project named 'CoolVision' with a Jira code of 'VISION'. It includes an 'ADD EFFORT' button. The bottom version is a form with input fields for 'Project name' and 'Jira code', each with a checkmark and an 'x' icon, and an 'ADD EFFORT' button.

Figure 5.2: Project details visualization

The first version of the screen includes the presence of a **Card** containing the basic information of the project. In particular, there are the name, the Jira code (both editable) and a button for adding any commercial/estimated working days for the current project. Obviously, in this first version of the app the two functions had not yet been implemented but the design began to be outlined which was then subsequently adapted for the next versions. In order to create a card of this type, the resolver associated with the reference query was implemented on the GraphQL side. In particular, it takes the ID of the associated project as input and extracts the information by invoking the **findById function**, offered by mongoose, in relation to the schema of the project being referenced:

```

1  const getProjectById = async function ({id_project}) {
2    const project = await ProjectSchema.findById(id_project);
3    if (!project)
4      throw new Error('Project not found ...');
5    return {
6      _id: project._id,
7      name: project.name,
8      jira: project.jira,
9      sold: project.sold.map(sold => {
10        return {
11          month: sold.month,

```



```

12         assignments: sold.assignments.map(assignment => {
13             return {
14                 _id_user: assignment._id_user,
15                 assigned: assignment.assigned
16             }
17         })
18     }
19 })
20 }
21 }

```

Listing 5.8: `getProjectById` in `./server/resolvers/resolver.ts`

5.1.3 Search for a project

At this point, the last feature relating to the search for a project, has been implemented through the use of a search bar. Every time it is typed, it changes a state that is used to filter on the individual rows of the table. Translated into code the implementation is as follows:

```

1 <Search>
2   <SearchIconWrapper><SearchIcon/></SearchIconWrapper>
3   <StyledInputBase
4     onChange={(e) => setSearch(e.target.value)}
5     placeholder="Search..."
6   />
7 </Search>
8 ...
9 projects.filter(
10   project => search === "" ? true :
11   project.name.toLowerCase().includes(search.toLowerCase()) ||
12   project.jira.toLowerCase().includes(search.toLowerCase())
13 ).map((project, idx) =>
14   <TableRow key={idx}>
15     <TableCell>{project.name}</TableCell>
16     <TableCell>{project.jira}</TableCell>
17     <TableCell>
18       <Button size="small" onClick={() =>
19         navigate(`/project/${project.id}`)
20       }>
21         Details
22       </Button>
23     </TableCell>
24   </TableRow>
25 )

```

Listing 5.9: `Projects.tsx` in `./client/src/components/List/Table`

5.1.4 Testing phase

During the testing phase prior to the official release of the first version, some usability issues were identified. The first issue was related to the **display of working hours** in the project overview page. Instead of displaying the progress of working hours based on the current day, it was deemed more useful to display it based on the month. This would allow for a quick and easy overview of the overall progress of work for the month without having to go through individual projects. The second issue was related to the Jira code field's fill during the project creation process. Indeed, **there was no check to ensure that the Jira code entered was valid** and corresponded to an existing project in the company's internal Jira server. To address this issue, a preliminary check was added to verify the Jira code before it was accepted and added as a project in the system. This would prevent the creation of projects with invalid Jira codes that do not correspond to any existing project in the server.

Both issues were fixed prior to the official release by implementing the necessary changes in the codebase. By taking user feedback seriously and proactively addressing usability issues, it was possible to deliver a more robust and user-friendly solution to the company's staffing needs. In particular, for the first problem it was decided to equip the modal with a **stepper**, in such a way as to break the process of inserting a new project into two phases. The first phase relating to the choice of the name and the second relating to the check of the inserted jira code, which will enable or not the creation, based on the presence or absence of the same code inserted on the jira server. The new component can be seen in the figure 5.3, while the code that made it possible to carry out this check is as follows:

The figure displays two screenshots of a project creation form, illustrating a two-step process using a stepper.

Top Screenshot (Successful Check):

- The stepper shows Step 1 (Insert Project Name) as completed (checkmark) and Step 2 (Insert Jira Code) as the current step (numbered 2).
- The "Jira Code *" input field contains the text "VISION".
- A blue "CHECK" button is visible next to the input field.
- Below the input field are "BACK" and "CREATE" buttons.
- A green success message box on the right states: "Project correctly present on Jira" with a green checkmark icon and a close button (X).

Bottom Screenshot (Unsuccessful Check):

- The stepper shows Step 1 (Insert Project Name) as the current step (numbered 1) and Step 2 (Insert Jira Code) as the next step (numbered 2).
- The "Project Name *" input field contains the text "CoolVision".
- A red error message box on the right states: "Project not present on Jira" with a red warning icon and a close button (X).
- Below the input field are "BACK" and "NEXT" buttons.

Figure 5.3: Components used related to the creation of a project. Steps and subsequent alerts in case of successful or unsuccessful check jira code

```

1  async function checkJiraCode(jira: string) {
2      return new Promise(async (res, _) => {
3          client.projects.getProject({projectIdOrKey: jira})
4              .then(_ => {
5                  res(true)
6              })
7              .catch(_ => {
8                  res(false)
9              });
10     })
11 }

```

Listing 5.10: checkJiraCode in ./server/src/dao.ts

For the second problem instead it was enough to change the display paradigm, in such a way that the object returned from the worklogs **referred to a specific day of the month passed as input**. In this way the visualization of both the AreaChart and the PieChart will refer to the month for which the data are being visualized and the progression on the X-axis of the Areachart refers to the different days of the month instead of the different hours of the day. The modified version can be seen in the figure 5.4



Figure 5.4: New screen relating to the list of projects with graphics associated with the monthly worklogs of the reference month

5.2 Management and assignment of the effort

Once the development, the test phase and the release of version 1 of the application were completed, it was carried the development of the features associated with the second version. Therefore, this section will indicate and explain the methodologies adopted for the development of features that allow to **add effort to a project** and the ability to **view these reference efforts**.

5.2.1 Adding effort to a project

As mentioned before, this second stage of app development is about adding commercial/estimated values for the projects. First of all, it is necessary to clarify what is meant by **commercial** and what by **estimated**. Commercial means the value relating to what has been sold (in terms of Man Day) for the development of a feature, or a list of features, for a single project in a certain established period of time. Estimated, on the other hand, means how much Man Days are estimated by the project managers and developers, in order to develop a certain functionality. After having clarified this, the first step was moved towards the screen that could allow the addition of this data for the single project.

The figure consists of two side-by-side screenshots of a web application interface for adding effort to a project.

The left screenshot shows the 'Effort configuration' step (Step 1 of 2). It includes a dropdown menu for 'Type *', input fields for 'Commercial total *' and 'Estimate total *' (both with 'MD Value' placeholder text), and date pickers for 'Start month' (03/2023) and 'End month' (03/2023). Navigation buttons 'BACK' and 'NEXT' are at the bottom.

The right screenshot shows the 'Month planning' step (Step 2 of 2). It features a table with columns 'Month', 'Commercial', and 'Estimate'. The table has rows for '03/2023' and '04/2023', each with input fields for 'MD Commercial val' and 'MD Estimate value'. A 'Total' row shows '0 / 7' for both columns. A message at the bottom states 'All estimate and commercial cells must be filled'. Navigation buttons 'BACK' and 'CREATE' are at the bottom.

Figure 5.5: Form associated with the addition of effort for a given project

The development was once again carried out using the Modal component made available to mui. In particular, an addition form was created which provides the possibility of choosing: the **type** of effort that is going to be added, that represents the macro-category of the functionality that will have to be developed, the **total commercial** and **total estimated** value (to be spread on months) and the **start and end month** to which the effort refers. Once again, all this is managed via a stepper, in such a way as to break up the insertion process, also in this case. In fact, the second step is related to the management of a dynamic table with three columns: **month**, **commercial** and **estimated**. Each row of this table refers to a month which is between the two previously selected months which act

as extremes. The purpose of this table is to enter values (for both the trade and estimated columns) for each month displayed. The form will automatically guide the user to enter the values with messages and on-screen highlights of the progress of the compilation. Of course nothing can be added until these errors/warnings are manually fixed by the user. An example associated with completing the form can be seen in the figure 5.5. Also in this case, after clicking on the Create button, a mutation on the GraphQL side is called, which refers to the following snippet:

```

1 mutation {
2   addSold (soldInput: {
3     _id_project: "${sold._id_project}",
4     type: "${sold.type}",
5     initial_month: "${sold.initial_month}",
6     final_month: "${sold.final_month}",
7     total_valued: ${sold.total_valued},
8     total_commercial: ${sold.total_commercial},
9     planning: [
10      ${sold.planning.map(p =>
11        '{
12          month: "${p.month}",
13          year: "${p.year}",
14          valued: ${p.valued},
15          commercial: ${p.commercial}
16        }'
17      ).join(",")}
18    ]
19  }) {
20    _id,
21    _id_project,
22    initial_month,
23    final_month,
24    type,
25    total_valued,
26    total_commercial,
27    planning {month, year, commercial, valued}
28  }
29 }
```

Listing 5.11: addSoldQuery in ./client/src/Query.ts

As can be seen from the composition of the mutation, the addition is handled through the use of an **input type (soldInput)**. The input type, in a GraphQL schema, is a special object type that groups a set of arguments together, and can then be used as an argument to another field. In the reference case, for adding effort, this type of input was created which expects to receive the parameters mentioned above. In particular, for the plan associated with the spread of commercial and estimated on individual months, it was decided to manage everything through an

array of **Planning-type objects**, which expects to receive as input: the reference month and year, the estimated value and the commercial value for that month and that specific project.

5.2.2 Monthly visualization of the effort

As regards the visualization of the monthly effort for each individual project, it was decided, also in this case, to use the **Table component** offered by mui, as can be seen from the figure 5.6. Each row of the table provides the presence of an accordion which, if opened, shows both through a graph and through another table, the trend for the individual months to which the effort refer, of the commercial and estimated values entered by the projects manager. In addition to this, a fixed row has been provided relating to the total, both as regards the commercial and as regards the estimated, of each effort present for that project. For the creation of the graph it was decided to use the **BarChart** component of the Recharts library, which is able to allow a visualization divided into two bars (commercial and estimated) for each month to which the effort refers. In this way it is possible to have a quantitative and qualitative view of the difference between commercial and estimated in each single month.

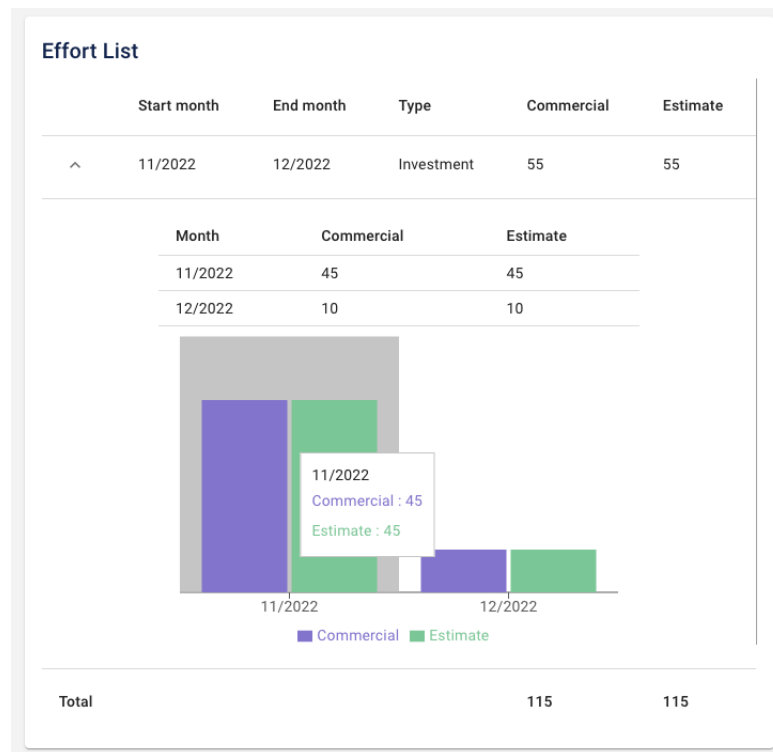


Figure 5.6: Effort list visualization for a single project

5.2.3 Testing phase

During this phase, the main objective was to identify any kind of usability problem in the developed features. Fortunately, no problems in these terms have been encountered, but it has been begun to think about how to make the sections of the application navigable through the use of a **Navbar**, in order to make any new pages related to subsequent features easily accessible. In fact, once this new need emerged, this new feature was included as a developable feature in version 6 of the app. In order to realize it once there is the complete picture of the various sections.

5.3 Staffing management of individual projects

In this sub-chapter it will be analyzed the development of the features related to the third version of the application. In particular, the focus will be on the possibility of assigning resources (employees) to a project in a given month and on the management of individual employees, in relation to the days available to be staffed, net of holidays/permits, in the various months of reference. Also in this case the aim will be to develop these features with a certain care regarding the user experience and the usability of the application.

5.3.1 Assigning resources to a project

This part of the application, had the objective of help project managers to assign working days (the so called "man days") to individual company employees, for each project present in the Coolvision database. To do this, it was created a monthly view, in which for each single project present, the number of estimated and commercial days for that project in that month, and the working days assigned to individual employees, were highlighted. The display screen can be seen in the figure 5.7.

The conception of the page is quite simple. It was decided to use a **MonthSlider custom component** for choosing the month with respect to view the data of the individual projects. It has been implemented using the **LocalizationProvider component** made available by the mui library. It provides for an interaction either directly with a click or through the appropriate arrows placed on the side. However, for the data, it was decided to use the mui **Table component** and it was organized into five columns:

1. The first provides for the presence of an arrow that could transform the single line into a real accordion, in such a way as to be able to display (later) additional information regarding the assignments

	Project	Commercial	Estimate	Assigned	
▼	CoolVision	45	45	12 MD	
▼	CnhApp	40	25	0 MD	
▼	Total	85	70	22	

Figure 5.7: Visualization screen of Commercial vs Estimated vs Assigned for projects in November 2022

2. The second provides an estimate of the commercial value entered for the single project in the reference month
3. The third provides an estimate of the estimated value entered for the single project in the reference month
4. The fourth provides an estimate of the assigned value entered for the single project in the reference month
5. The fifth provides for the presence of a component that allows, if clicked, to carry out the individual assignments

To ensure that the second and third columns are filled, the **getSoldsByMonth** query has been implemented. It returns an array of objects that include the reference project, the type of value associated with the commercial/estimated for that project, and the total commercial and estimated value. All these data are obviously related to the month passed as input to the query. The code is the one as following:

```

1  const getSoldsByMonth = async function ({month}: MonthProps) {
2    const solds = await SoldSchema.find()
3    let soldsByMonth: {
4      project: any,
5      type: String;
6      total_valued: number;
7      total_commercial: number;
8    }[] = []
9    solds.forEach(sold => {
10      sold.planning.forEach(plan => {

```



```

11         if (plan.month === month.split("/")[0]
12         && plan.year === month.split("/")[1])
13             soldsByMonth.push({
14                 project: getProjectInformationById({
15                     id: sold._id_project
16                 }),
17                 type: sold.type,
18                 total_valued: plan.valued,
19                 total_commercial: plan.commercial
20             })
21     })
22 })
23 return soldsByMonth
24 }

```

Listing 5.12: getSoldsByMonth in ./server/resolvers/resolvers.ts

In addition to this, the creation of an extra row in the table was added in order to enclose the total of the values mentioned up to this moment, of all the projects in that single month.

5.3.2 Use of DataGrid for assigning working days

At this stage of the development, to realize the possibility of assigning working days to individual employees, the solution shown in the figure 5.8 was devised. In particular, everything was managed through the use of a **Modal**, inside which a table was inserted in the form of a **DataGrid component** made available by mui. The management of the data present inside was carried out through the use of an API that was able to fetch the data associated with individual employees from the **CoolPlatform application**, that is the web application used by the company to record the associated data to employees. Therefore, the first step was to expose, on the **CoolPlatform** side, a rest API that was able to return such data.

Before going into this part of the application development, it is necessary to introduce the web application called **CoolPlatform**. It represents an internal application used by Coolshop employees to monitor the status of their progress (in terms of personal and corporate Objective and Key Results OKRs), have access to training material related to the various workshops held by company members on various topics, and verify the status of the days of holidays/permits that can be requested and/or have been already requested. Through this application, an attempt has been made to expose a rest API that was able to return the information of all the employees present with the addition of the number of days of vacation/permits requested by them and correctly approved, in the month in which they were reporting. The project in question, on the back-end side, uses **Symfony** [22] as a framework, and its controllers, to expose the APIs of interest. Symfony

represents a "set of reusable PHP components and a PHP framework for web projects". It has been used in CoolPlatform for the development of all the back-end part of the application, especially thanks to the use of its controllers. In Symfony, a controller is usually a class method which is used to accept requests, and return a Response object. When mapped with a URL, a controller becomes accessible and its response can be viewed. To facilitate the development of controllers, Symfony provides an Abstract Controller. To expose the API of interest, a special controller was created in CoolPlatform (called **ExternalController**) in which the **retrieveUsersHolidays** method was implemented, which is as follows:

```

1 public function retrieveUsersHolidays($month) {
2     $response = new JsonResponse();
3     if(!$this->checkAuthorization()) {
4         $response->setStatusCode(403);
5         $response->setContent(
6             "Authentication failed (Bearer token)"
7         );
8     } else {
9         $em = $this->getDoctrine()->getManager();
10        $usersHolidays = [];
11        $users = $em
12            ->createQueryBuilder()
13            ->select('u.id', 'u.firstName', 'u.lastName',
14                'u.googleProfilePhoto', 'd.name as role')
15            ->from('App:User', 'u')
16            ->innerJoin(
17                'u.department', 'd', 'WITH', 'u.department = d.id'
18            )
19            ->getQuery()->getResult();
20        foreach ($users as $user){
21            $requestsHoliday = $em
22                ->createQueryBuilder()
23                ->select(
24                    "h.startDateTime", 'h.endDateTime',
25                    'h.status', 'h.type'
26                )
27                ->from('App:HolidayRequest', 'h')
28                ->innerJoin(
29                    'h.user', 'u', 'WITH', 'h.user = :userId'
30                )
31                ->where(
32                    "h.startDateTime between :start and :end"
33                )
34                ->orWhere(
35                    "h.endDateTime between :start and :end"
36                )
37                ->setParameter(

```

```










38         'start', date('Y-m-d', strtotime($month))
39     )
40     ->setParameter(
41         'end', date('Y-m-t', strtotime($month))
42     )
43     ->setParameter('userId', $user['id'])
44     ->getQuery()->getResult();
45     $userHoliday = new ExternalHoliday();
46     $userHoliday->user = $user;
47     $userHoliday->holidays = $requestsHoliday;
48     array_push($usersHolidays, $userHoliday); }
49     $response->setContent(json_encode($usersHolidays));
50 }
51 return $response;
52 }

```

Listing 5.13: retrieveUsersHolidays in CoolPlatform controller

Staffing for CoolVision - 11/2022

☐ Show only available 🔍 Search...

Photo	Name	Role	Project assigned	Monthly Assigned	Remaining
	Stefano Rainò	Hero	Assign 0	0	22
	Enrico D'Oro	Hero	Assign 9	9	13
	Federica Giglio	Paladin	Assign 17	17	5
	Valentina Ieraci	S.H.I.E.L.D.	Assign 0	0	22
	Marcello Zampella	Paladin	Assign 0	0	22
	Alberto Piras	Hero	Assign 19	19	-1
	Riccardo Andretta	Paladin	Assign 0	0	22
	Marco Zangara	Paladin	Assign 0	0	22
	Andrea Vincelli	Jedi	Assign 0	0	22

CLOSE SAVE

Figure 5.8: Table related to the possible assignments for the CoolVision project for November 2022

As can be seen, the exposure of the API was carried out with a protection mechanism linked to a **Bearer token**, so that the data returned by it can only be accessed via authentication. In addition to this, it is possible to verify how the management of the database query was carried out through the use of **Doctrine** [23]. Doctrine ORM is an **object-relational mapper** (ORM) for PHP 7.1+ that

provides transparent persistence for PHP objects. It uses the Data Mapper pattern at the heart, aiming for a complete separation of the domain/business logic from the persistence in a relational database management system. In this case it was used to create the query that allows to return the data of interest from CoolPlatform.

At this point, the assignment of working days was done through the use of a component internal to the display table. In particular, first, through a **useEffect**, the array of users was created by referring to the **getUsersHolidays** API. In this array, for each user present, the following information is stored: the id, the name, the number of days of leave/vacation approved for the month in question, the photo and the role. The **useEffect** is the following:

```

1  useEffect(() => {
2    if (month)
3      getUsersHolidays(month).then(users => {
4        let newUsers: User[] = []
5        users.forEach(user => {
6          newUsers.push({
7            _id: String(user.user.id),
8            name:
9              user.user.firstName + " " +
10             user.user.lastName
11          },
12          holidays: user.holidays.map((holiday: any) => {
13            if (holiday.status === 'APPROVED_BY_HR') {
14              if (month.startOf('month') >
15                dayjs(holiday.startDateTime.date))
16                return
17              - month.startOf('month')
18                .diff(dayjs(holiday.endDateTime.date)
19                  .format('YYYY-MM-DD'), 'day')
20            } else {
21              if (month.endOf('month') <
22                dayjs(holiday.endDateTime.date))
23                return month.endOf('month')
24                .diff(dayjs(holiday.startDateTime.date)
25                  .format('YYYY-MM-DD'), 'day')
26            } else
27              if (dayjs(holiday.endDateTime.date)
28                .diff(dayjs(holiday.startDateTime.date),
29                  'hour') > 0)
30                return
31                Number(((dayjs(holiday.endDateTime.date)
32                  .diff(dayjs(holiday.startDateTime.date),
33                    'hour')/8).toFixed(2))
34              )
35              return 1
36          })
37      })
38  }

```

```

37         }
38         return 0
39     }).reduce((day1: any, day2: any) => day1 + day2, 0),
40     photo: user.user.googleProfilePhoto,
41     role: user.user.role,
42   })
43 })
44   setUsers(newUsers)
45 })
46 }, [month])

```

Listing 5.14: useEffect in AssignedModal.tsx

In this way the users are returned correctly and consequently, another useEffect was used to render the individual rows of the table which carries out the calculation associated with the monthly assignments for the individual user and the monthly working days for the reference month taking into account: Italian public holidays in that month and permits/holidays correctly approved for the individual user. Translated into code it is as follows:

```

1  useEffect(() => {
2    if (users && assignments && month && monthAssignments)
3      setRows(
4        users.map(user => {
5          return {
6            id: user._id,
7            Name: user.name,
8            Role: user.role,
9            Photo: user.photo,
10           'Monthly Assigned':
11             monthAssignments.find(
12               assignment => assignment._id_user === user._id
13             ) ?
14             monthAssignments.find(
15               assignment => assignment._id_user === user._id
16             ).assigned : 0,
17           'Project assigned':
18             assignments?.find(assignment =>
19               assignment.user.id === user._id
20             ) ?
21             assignments.find(assignment =>
22               assignment.user.id === user._id
23             ).assigned : 0,
24           Remaining:
25             monthAssignments.find(assignment =>
26               assignment._id_user === user._id
27             ) ?
28             getWeekdaysInMonth(

```

```

29         Number(month?.get('year')),
30         Number(month?.get('month'))
31     ) -
32     Number(monthAssignments?.find(assignment =>
33         assignment._id_user === user._id
34     )?.assigned) -
35     Number(user.holidays) :
36     getWeekdaysInMonth(
37         Number(month?.get('year')),
38         Number(month?.get('month'))
39     ) -
40     Number(user.holidays),
41     }
42 })
43 )
44 }, [users, assignments, month, monthAssignments])

```

Listing 5.15: useEffect in AssignedModal.tsx

As can be seen, the calculation was carried out taking into consideration the so-called **monthAssignments**, i.e. the assignments in that specific month, filtered by the reference user ID, and the simple **assignments** instead which refer to the assignments for that project of all users present. Once this is done, the only thing missing is to create the TextField component in an interactive way that is able to memorize the changes for each individual user in real time. It was made as follows:

```

1  {
2      field: "Project assigned",
3      width: 180,
4      filterable: false,
5      sortable: false,
6      getApplyQuickFilterFn: undefined,
7      renderCell: (params: any) => {
8          return (
9              <TextField
10                 value={params.row['Project assigned']}
11                 variant='standard'
12                 type='number'
13                 size='small'
14                 label='Assign'
15                 error={params.row.Remaining < 0}
16                 onChange={(e) => {
17                     if (Number(e.target.value) >= 0) {
18                         let newRows = rows.map(row => {
19                             if (row.id === params.row.id) {
20                                 if (Number(row.Remaining) -
21                                     (Number(e.target.value) -
22                                     Number(row['Project assigned'])) < 0)

```

```

23         setDisableSave(true)
24     else
25         setDisableSave(false)
26     return {
27         ...row,
28         'Project assigned':
29             Number(e.target.value)
30     },
31     'Monthly Assigned':
32         Number(row['Monthly Assigned']) +
33         Number(e.target.value) -
34         Number(row['Project assigned'])
35     },
36     Remaining:
37         Number(row.Remaining) -
38         (Number(e.target.value) -
39         Number(row['Project assigned']))
40     }
41 } else
42     return row
43 })
44 setRows(newRows)
45 }
46 }}
47 />
48 )
49 }
50 },

```

Listing 5.16: renderCell in AssignedModal.tsx

5.3.3 Testing phase

During the testing phase of the feature relating to assignment, there were no significant issues identified. On the contrary, the implementation was well received, and project managers appreciated the ability to manage data using accordion components. As a result of the positive feedback, it was decided to expand the functionality to enable users to compare the current month's assignments with the previous/next month's assignments directly from the same page. The aim of this enhancement is to provide users with a quick and easy way to compare assignment data without having to manually scroll through multiple months. The extended feature will allow users to toggle between months and view a side-by-side comparison of the assignment data for the selected months. The expanded functionality will require minor modifications to the existing code. The code will need to be updated to include a new toggle button that allows users to switch between the current month and the previous/next month. The data for the selected

months will be displayed in a new section of the page, enabling users to view the comparison side by side. The result can be seen in the figure 5.9

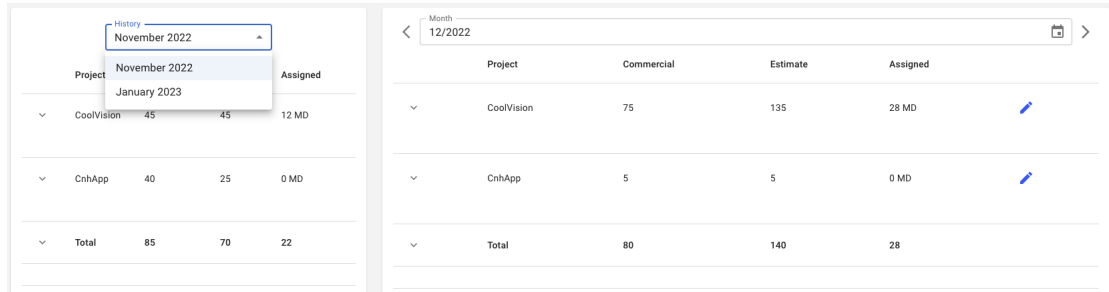


Figure 5.9: Visualization screen of Commercial vs Estimated vs Assigned for projects with the new section for previous/next month

In conclusion, the testing phase for the assignment feature was successful, with positive feedback received from users. The decision to extend the feature to include the comparison functionality was made based on the feedback received, with the aim of improving user experience and streamlining the assignment management process.

5.4 Worklogs and Staffing visualization

In this chapter it will be explained the development of version 4 of the application, which includes features for visualizing statistics related to worklogs and staffing for individual projects. This chapter will delve into how the charts were designed and how they were integrated into the application's interface. The ability to visualize worklogs and staffing data is essential for monitoring project progress and identifying areas that require attention. Therefore, in this version, the focus is on enhancing the application's ability to provide users with a clear and concise view of project-related statistics. To achieve this goal, various charts and graphs have been designed to display the data in a user-friendly manner. The charts include bar charts, radar chart, and pie charts, each designed to showcase different aspects of the data. The color schemes and data representation were carefully chosen to ensure that the charts are easy to read and interpret. Moreover, these charts have been integrated into the application's interface, making it simple for users to navigate through the various features and access the data they need.

5.4.1 Visualization of the history of the worklogs

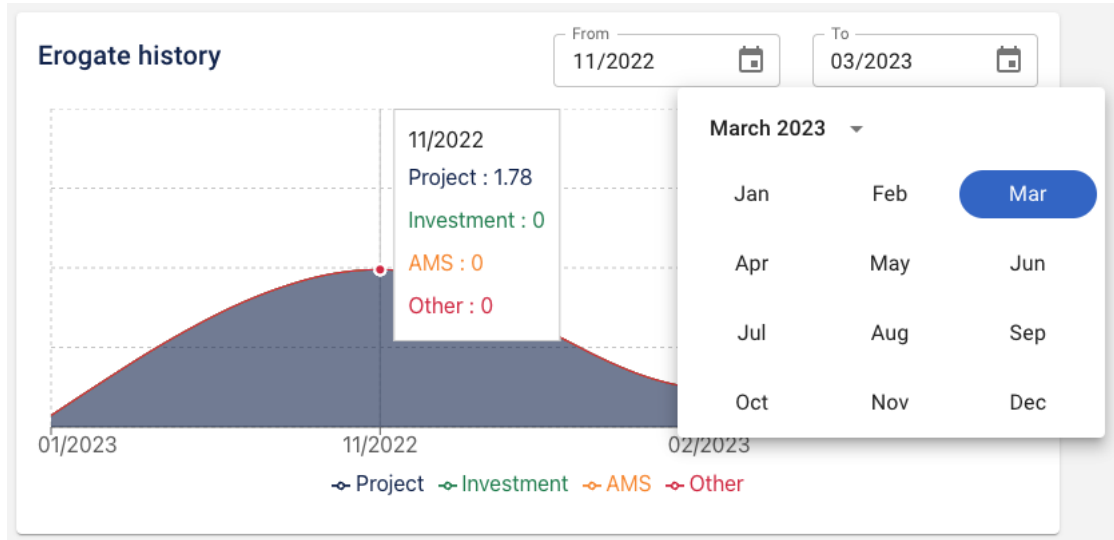


Figure 5.10: Graph showing the history of working hours paid for a specific project divided into categories

In order to develop the data visualization part, the Recharts library was once again used. As thanks to it, it was possible to create an area chart that showed the days of work provided in the individual months divided by category. The graph created can be seen in the figure 5.10. The categories of interest used to track employee working days, are the same as the categories that can be selected when effort is added to the project, with the addition of the "Other" category which refers to all the worklogs that refer to differently labeled projects present on Jira. Also in this case, the `getJiraWorklogs` API was used to return the worklogs, which however restricts the selection only and exclusively to a start and end date passed as a parameter. In fact, the visualization has been designed in such a way as to keep track of the worklogs on the basis of two months (initial and final) which can be selected through the use of two components of the `DatePicker` type. By default, the display is restricted with respect to the current month and the four months preceding it, but in this way, using the `DatePickers`, it is possible to view the history of the worklogs based on two initial and final dates chosen as desired by the user. The choice of the place in which to insert this graphic fell on the project detail page. In such a way as to be able to add more information regarding the history of the worklogs, together with the history of the effort assigned to the project in question. In this way it is possible to have a direct and specific comparison relating to the single project

5.4.2 View staffing for a project in the current and previous/next month

Once the graph associated with the history of the worklogs has been created, what is missing to conclude this version is the visualization of the staffing for a particular project (assigned through the previously mentioned features). The idea behind the development was to exploit the accordion created in relation to the single rows of the table mentioned in section 5.3.1. In this way, by expanding the row, it would be possible to view a whole series of statistics relating to the project in question in the reference month. The final result associated with the realization of this feature can be seen in the figure 5.11. When the accordions are opened, they show the following data:

- Number of man days related to the commercial and estimated value, for the selected month and for the project in question, divided by category (AMS, Project, Investment)
- List of employees assigned to that project in the month in question (divided by role) with related number of days assigned in that month for that project

In addition to this, it was decided to add a **radar chart**, for the month displayed on the right, which can be used to have a more immediate view of how the commercial and estimated values are distributed in the month in question for that project.

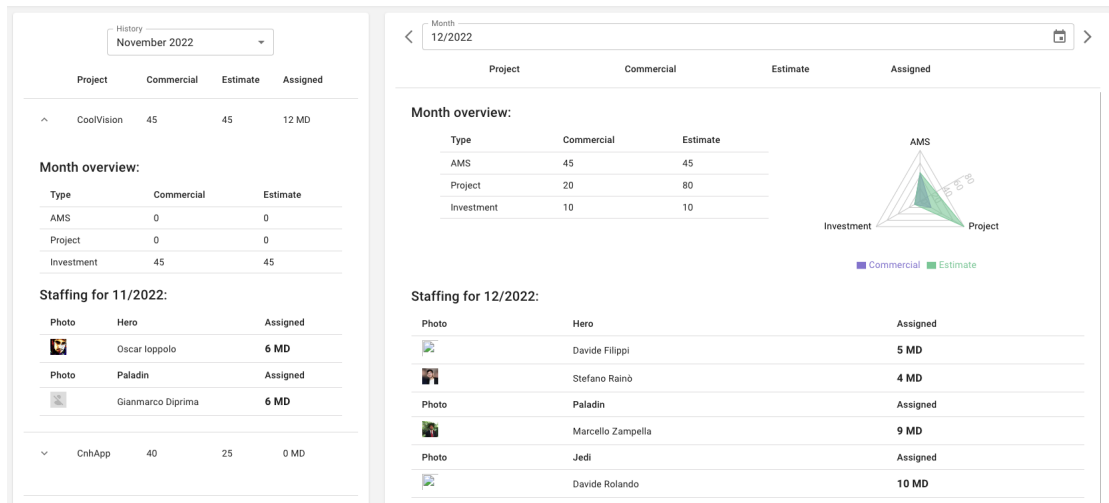


Figure 5.11: Visualization of assignments on individual projects in the reference months

To achieve all this, first of all it is necessary to understand how the data associated with assignments and planning on the MongoDB side are stored. As can be

seen from the image 5.12, for each stored project, the "sold" is an array of objects containing the reference month and a list of assignments, which contain the id of the user in question and what has been assigned for that user specific in the reference month of the assignment. In this way it is possible to keep track of the assignments made on the specific project for each month and consequently, when assignments are added, this array will be modified in relation to the month to which the application is referring. However, for the "planning", what happens is that a schedule is stored for each single sales added. The schedule is an array of objects that store: the month referred, the commercial and estimated value associated with that specific month for that specific project and the specific category of total sales. In this way it is possible to keep track of the categories of single sales associated with the projects and the months with respect to which they refer, and also it is possible to keep track of the assignments for a specific project in a given month. Once this was explained it was necessary to expose a new rest api on the CoolPlatform side that could return the information of the employees given the id. It is the following:

```

1 public function retrieveUsersByIdIn() {
2     $response = new JsonResponse();
3     if(!$this->checkAuthorization()) {
4         $response->setStatusCode(403);
5         $response->setContent(
6             "Authentication failed (Bearer token)"
7         );
8     } else {
9         $body = json_decode(file_get_contents('php://input'));
10        if (empty($body)) {
11            $response->setContent("Users ids not found");
12            $response->setStatusCode(404);
13        } else {
14            $users = [];
15            $em = $this->getDoctrine()->getManager();
16            foreach ($body as $user_id) {
17                $user = $em->createQueryBuilder()
18                    ->select(
19                        'u.id',
20                        'u.firstName',
21                        'u.lastName',
22                        'u.googleProfilePhoto',
23                        'd.name as role'
24                    )
25                    ->from('App:User', 'u')
26                    ->innerJoin(
27                        'u.department', 'd', 'WITH',
28                        'u.department = d.id'

```

```

29         )
30         ->where('u.id = :userId')
31         ->setParameter('userId', $user_id)
32         ->getQuery()
33         ->getOneOrNullResult();
34         if($user)
35             array_push($users, $user);
36     }
37     $response->setContent(json_encode($users));
38 }
39 }
40 return $response;
41 }

```

Listing 5.17: retrieveUsersByIdIn in ./src/Controller/ExternalController.php

```

{
  "_id": "636430a50b1e0e09363a059",
  "_id_project": "63642d460b1e0e09363a00f",
  "initial_month": "11/2022",
  "final_month": "12/2022",
  "type": "Investment",
  "total_valued": 55,
  "total_commercial": 55,
  "planning": [
    {
      "month": "11",
      "year": "2022",
      "valued": 45,
      "commercial": 45,
      "_id": "636430a50b1e0e09363a05a"
    },
    {
      "month": "12",
      "year": "2022",
      "valued": 10,
      "commercial": 10,
      "_id": "636430a50b1e0e09363a05b"
    }
  ],
  "__v": 0
}

```

```

{
  "_id": "63642d460b1e0e09363a00f",
  "name": "CoolVision",
  "jira": "VISION",
  "sold": 23,
  "month": "11/2022",
  "assignments": [
    {
      "_id_user": "95",
      "assigned": 6,
      "_id": "636445567478485b30499b2a"
    },
    {
      "_id_user": "89",
      "assigned": 4,
      "_id": "636445567478485b30499b2b"
    },
    {
      "_id_user": "87",
      "assigned": 6,
      "_id": "636445567478485b30499b2c"
    },
    {
      "_id_user": "42",
      "assigned": 6,
      "_id": "636445567478485b30499b2d"
    },
    {
      "_id": "63643e4efdf4991e94d1a4d0"
    }
  ],
  "__v": 1
}

```

Figure 5.12: Data stored on the MongoDB side regarding the CoolVision project and the sales referring to CoolVision

As can be seen, the API takes as input from the body a list of ids, which are the ids associated with individual users that were stored on the MongoDB side when assignments were made for a given project in a given month. It returns in response all the information concerning the user in question starting from the profile photo of the google account and ending with his department (the so-called "role"). Therefore, in this way, it was easy to start from the assignments made for that project and build the user table through the following snippet:

```

1 roles.map((role, idx) => {
2     if (assignments?.find(assignment =>
3         assignment.user.role === role)
4     )
5     return (
6         <React.Fragment key={idx}>
7             <TableHead>

```

```

8      <TableRow>
9          <TableCell size="small">Photo</TableCell>
10         <TableCell>{role}</TableCell>
11         <TableCell>Assigned</TableCell>
12     </TableRow>
13 </TableHead>
14 <TableBody>
15 {
16     assignments.filter(assign =>
17         assign.user.role === role
18     ).map((assignment, key) => {
19         return (
20             <TableRow key={key}>
21                 <TableCell size="small">
22                     <Icon>
23                         <img
24                             className='image'
25                             alt=""
26                             src={assign.user.photo}
27                             height={25} width={25}
28                         />
29                     </Icon>
30                 </TableCell>
31                 <TableCell>
32                     {assign.user.name}
33                 </TableCell>
34                 <TableCell>
35                     <Typography>
36                         <b>
37                             {assignment.assigned+" MD"}
38                         </b>
39                     </Typography>
40                 </TableCell>
41             </TableRow>
42         )
43     })
44 }
45 </TableBody>
46 </React.Fragment>
47 )
48 else
49     return (<React.Fragment key={idx}/>)
50 })

```

Listing 5.18: Table with assignments for a specific project

5.4.3 Testing phase

During the testing phase for the worklogs and staffing visualization features, it was not encounter any significant issues. The features developed were well-received by project managers, and no major bugs were identified. However, a request was made to enrich the project detail page with a summary chart that could provide an overview of the commercial and estimated totals for the project in question. This chart would enable users to get a complete picture of the project statistics directly from the project detail page. To fulfill this request, it was chosen to implement a PieChart as the summary chart. This chart was designed to show the percentage breakdown of the commercial and estimated totals for the project in question. By integrating this chart into the project detail page, users can quickly and easily see the overall project statistics and make informed decisions based on this data. The screen of the new detail page can be viewed in the figure 5.13

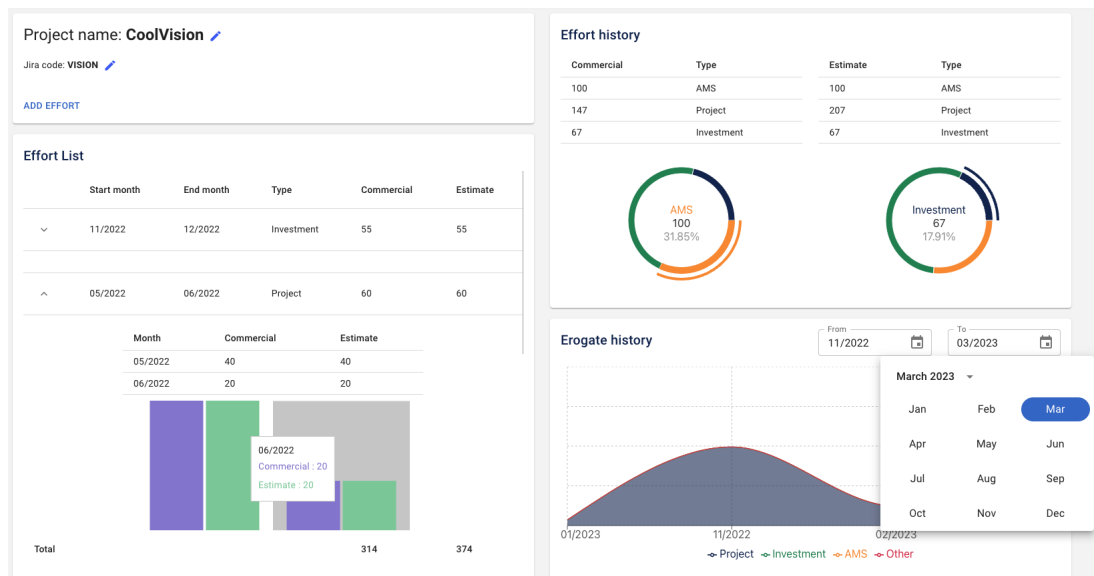


Figure 5.13: New project details view with piecharts for commercial and estimate statistics

In conclusion, the testing phase for the worklogs and staffing visualization features was successful, with no major issues identified. The request to add a summary chart to the project detail page was a valuable suggestion, and it could add significant value to the overall functionality of the application.

5.5 Visualization of report statistics

In this section, it will be explained how was implemented a new screen in the application that allows users to visualize the monthly worklog trends for each individual project. Similar to the staffing visualization feature 5.4.2, this screen presents these trends through charts that indicate the workload status, based on the estimated and commercial values associated with the project for each specific month. The aim of this new feature is to provide users with a comprehensive view of the project's worklog statistics, allowing them to analyze and interpret the data in a meaningful way. This visualization will enable users to identify the workload distribution across different months and compare it to the commercial and estimated values, helping them make informed decisions about project management and resource allocation. This feature, as the other, is designed to be intuitive and user-friendly, with clear and easy-to-understand charts that enable users to quickly and easily interpret the data. With this new screen, users will be able to identify any potential workload imbalances and take proactive measures to address them.

Overall, the visualization of report statistics is a valuable addition to the application, providing users with a powerful tool for project management and decision-making. In the following sections, the implementation of this feature will be detailed, including the design and integration of the charts, and the testing and refinement process.

5.5.1 Consultation of Worked vs Estimated monthly for each project

In order to develop this type of screen, detailed above, it was decided to start from a view similar to 5.4.2. In this case, however, what is highlighted is the comparison between what has been estimated for the commercial value associated with that project in that particular month, and the effective worked days. This is why what can be seen in the figure 5.14 was conceived. In particular, it can be seen how the conception and construction of the table is very similar with the addition of a progress bar that highlights the trend of hours worked, which turns red if these hours exceed those estimates for the commercial in that specific month for that project. Then, it is possible to immediately have visual feedback on the progress of the work associated with a project. In addition to this, a table display of the division of hours associated with sales and hours worked on the basis of the 4 categories of interest has been provided. The same 4 categories featured in the two graphs below:

- The first represents a transposition of the previously described table, and has been implemented in order to better highlight the differences that there may

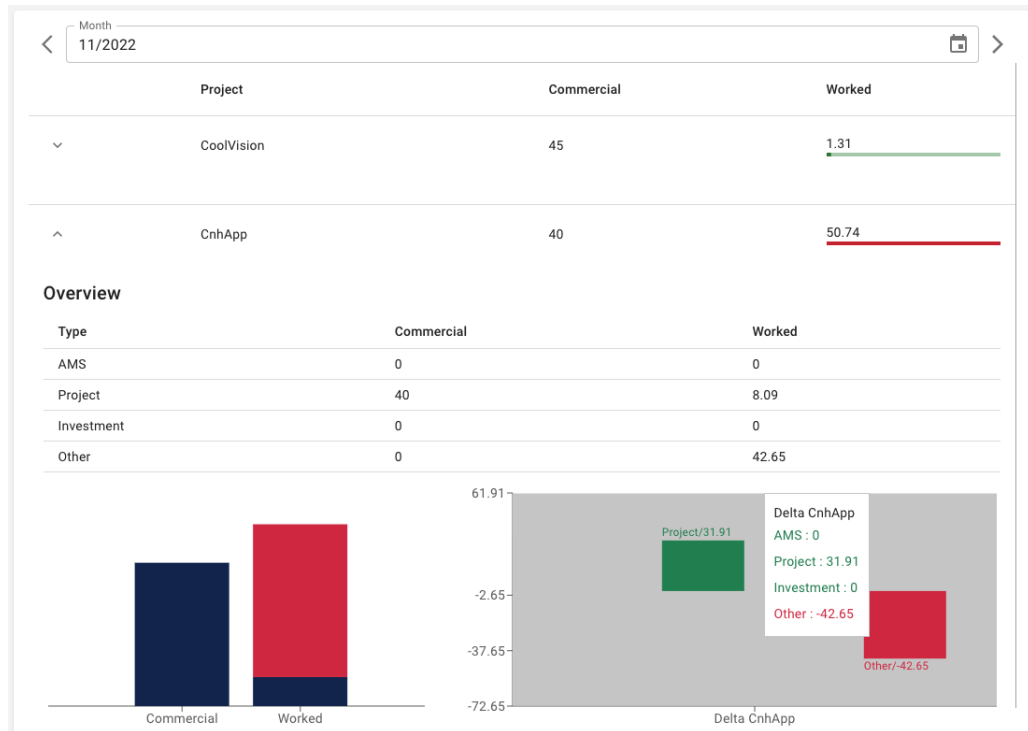


Figure 5.14: Report of CnhApp project for November 2022

be between the four categories of interest both as regards the commercial and the worked

- The second represents a graph whose purpose is to highlight the delta between commercial and worked relative to the individual categories in the month of interest. It is a bar chart in which each bar represents the difference between commercial and worked for each individual category. If the difference is greater than zero the bar will be green and pointing up, otherwise it will be red and pointing down

In this section, if there were working hours associated with projects not present in the database, it was decided to add the **Missing projects** item to the table, which could include these hours worked in the month in question. This is because the purpose of this screen is to try to highlight as much as possible to the project managers the progress of the work and above all the actual correspondence with what was estimated when adding the sales of the individual projects. In such a way that if some of them are not present, it can be checked and remedied immediately. Also in this case, the display has been extended so that the statistics relating to the previous/next month of the individual projects can also be included on the same

page. Unlike what was done before, however, the visualization has been restricted to only the graph relating to the comparison between worked and commercial in order not to burden the page with data but still allow for a more complete overview of the progress of the projects. The page can be viewed in the figure 5.15

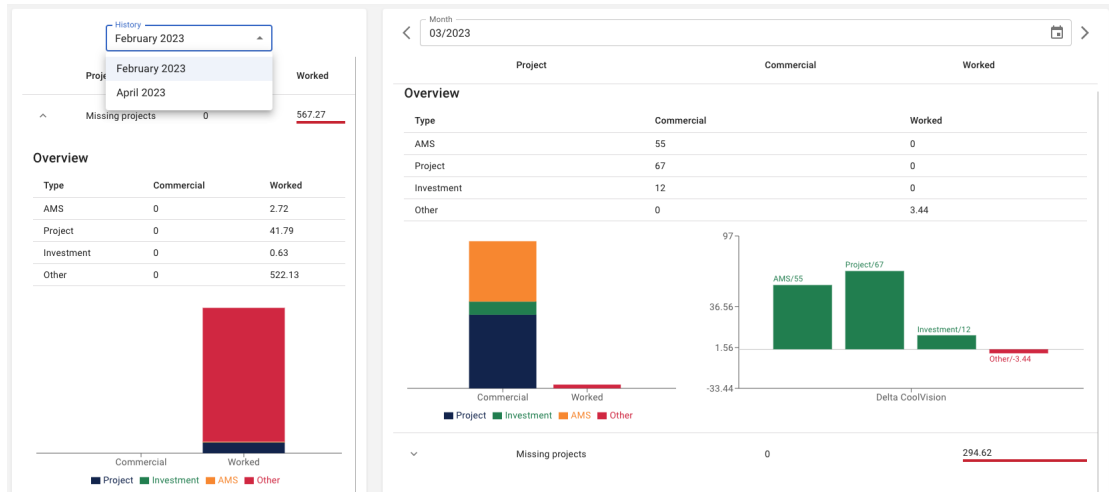


Figure 5.15: Report page with statistics regarding previous/next month

5.5.2 Testing phase

During the testing phase of this new feature released with the new version of the application, there were no major problems encountered. In particular, the development was focused on the aim of making the screen consistent with the screens previously presented in such a way as to maintain a certain flow of actions that can be easily understood by the end user. Therefore, the remaining days before the start of development for the next release were used to optimize parts of the code in order to make the use of content more fluid in terms of time, given the large amount of data to be viewed and handle in single screens.

5.6 Login and security management with last updates

The Login and Security Management chapter focuses on the implementation of a secure and controlled login process for the application. In this chapter, it will be explained how the login process was designed and implemented, utilizing Google APIs and restricting access to the application to only those with a company email address ending in "@coolshop.it". Moreover, it was also implemented the possibility

to update an existing project, in order to change the settings like name and jira code according to an eventually server-side jira modification. And obviously the new feature concerning the presence of a Navbar, that can act as a navigation menu between the various pages of the app, as described in 5.2.3

5.6.1 Login via Google

As regards the last phase of the development, what was wanted to achieve was an authentication system that would allow access, via Google, to accounts belonging to the internal domain of Coolshop. To do this, first of all, a separate screen was created that could act as a login interface. It can be seen in the figure 5.16, and it is a very simple interface, in fact it includes: the logo of the app and the login button. The login button was developed using the **react-google-login** library, which provides a component, namely `GoogleLogin`, which represents a real button with specific props for managing login through google services. In particular, the exploited props are two:

1. **clientId**: Represents an identifier created through the Google Developers platform which allows the application in question to be able to use google services in an appropriate manner (in this case the login service)
2. **onRequest**: Represents a method that identifies what needs to be done once the button has been clicked

In this case, what is triggered on button click is the redirection to the `/auth/-google` route which is fetched in the back-end via express, using **Passport** [24] as middleware. Passport is authentication middleware for Node.js. Extremely flexible and modular, Passport can be unobtrusively dropped in to any Express-based web application. A comprehensive set of strategies support authentication using Google services and more. In particular, passport in this case has been configured in such a way that it can accept logins via google of accounts associated with the Coolshop domain, and this type of filtering has been developed as follows:

```
1 passport.use(new GoogleStrategy(  
2   {  
3     clientId: env.google.clientID,  
4     clientSecret: env.google.clientSecret,  
5     callbackURL: "/auth/google/callback",  
6   },  
7   function (accessToken, refreshToken, profile, cb) {  
8     profile._json.hd === "coolshop.it" ?  
9       cb(null, profile)  
10    :  
11  })
```

```
11         cb(null, false);
12     }
13 })
```

Listing 5.19: Passport strategy in `./server/src/index.ts`

As for all the other routes that can be fetched in case of login success/failure or other, they have been enclosed in a separate file called `./server/src/route.ts` and are as follows:

```
1  router.get("/login/success", (req, res) => {
2      if (req.user) {
3          res.status(200).json({
4              success: true,
5              message: "Login successful",
6              user: req.user,
7          });
8      }
9  });
10
11 router.get("/login/failed", (req, res) => {
12     res.status(200).json({
13         success: false,
14         message: "Login failed",
15         user: null
16     });
17 });
18
19 router.get("/logout", (req, res) => {
20     req.logout((err) => console.log(err));
21     res.redirect(CLIENT_URL);
22 });
23
24 router.get("/google", passport.authenticate("google", {
25     scope: [
26         "profile",
27         "https://www.googleapis.com/auth/userinfo.email"
28     ]
29 }));
30
31 router.get(
32     "/google/callback",
33     passport.authenticate("google", {
34         successRedirect: CLIENT_URL,
35         failureRedirect: CLIENT_URL,
36     })
37 );
```

Listing 5.20: Login route in `./server/src/route.ts`

As can be seen, in case of successful login, the application will redirect to the main page of the app which in turn, through a `useEffect`, will verify that the fetch has been successful and will change the login status to true in such a way that all the routes of the application can be accessed correctly. The `useEffect` is as follows:

```

1  useEffect(() => {
2    fetch('
3      ${env.server.scheme}://${env.server.host}:
4      ${env.server.port}/auth/login/success',
5      {
6        method: "GET",
7        credentials: "include",
8        headers: {
9          "Accept": "application/json",
10         "Content-Type": "application/json",
11         "Access-Control-Allow-Credentials": "true"
12       },
13     })
14   ).then((response) => {
15     if (response.status === 200)
16       return response.json();
17   })
18   .then((_) => {
19     setLogin(true)
20   })
21 }, [])

```

Listing 5.21: `useEffect` in `./client/src/App.tsx`

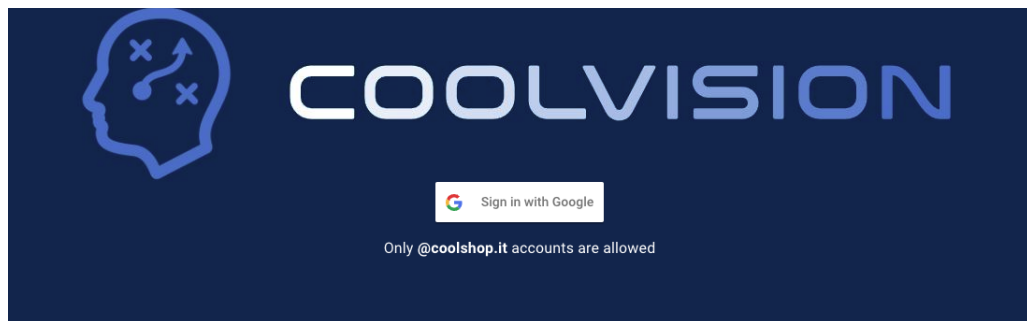


Figure 5.16: Login interface

5.6.2 Update a project

For these phase of the project instead, it is developed a feature left open in previous versions. In particular, the possibility of modifying the personal data of the project,

i.e. name and Jira code, has been added within the project detail screen. The feature has been developed in such a way that once the click on the pencil icon is triggered, the screen that can be viewed in the figure is visualized. Once this is done, it will be possible to correctly modify the name and/or the jira code and save these modifications by clicking on the tick. The code that will make the modification to the database is the following:

```
1 const updateProject = async function ({id_project, name, jira}) {  
2   const project = await ProjectSchema.findById(id_project);  
3   if (!project)  
4     throw new Error('Project not found ...');  
5   project.name = name;  
6   project.jira = jira;  
7   return await project.save();  
8 }
```

Listing 5.22: updateProject in ./server/resolvers/resolvers.ts

5.6.3 Navbar navigation menu

This section is dedicated to the development of the latest feature of the project, which is a Navbar able to guide the user between the various pages of the application. To do this it was decided to use the **MuiAppBar** component made available by mui. In particular, it has been drawn up in a personalized way in order to be used together with the **MuiDrawer** component, always made available by mui. In this way it is possible to manage a side menu including 5 different menu items, namely: Projects, Report, Plan, Add Project and Logout. The first refers to the screen associated with the list of projects, and represents the home page of the application. The second refers to the page including the list of assignments made on individual projects for each reference month. The third instead refers to the screen including the statistics relating to the work carried out in the single months compared with the actual commercial sold in those months. The other two menu items, on the other hand, have been separated from the other three items by a divider. In particular, it was decided to insert the Add Project menu item in the personalized drawer in such a way as to be able to add a project in any application context in which one is located, without necessarily having to be positioned in a specific page to carry out this creation. The style of the navbar with the side menu can be seen in the figure 5.17

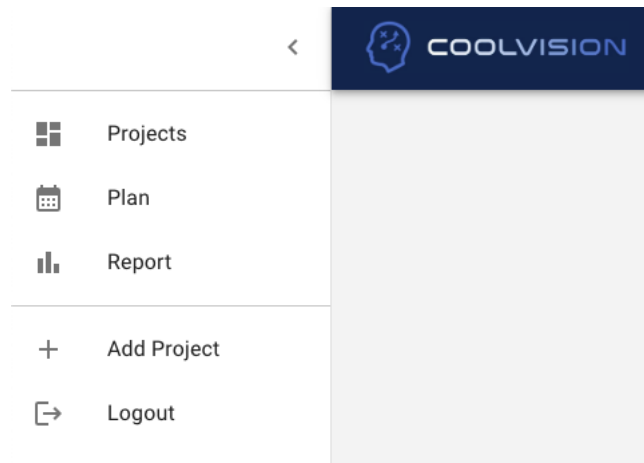


Figure 5.17: Navbar and drawer in coolvision web app with logo of the application

5.6.4 Testing phase

During the testing phase of the project, as always, all the features are tested to ensure their functionality and usability. It was faced a minor issue related to the JIRA code verification process. When a change is triggered to the code, is needed to verify the JIRA code associated with the change. While performing this verification, it was found that the insert of JIRA code was sometimes inconsistent with the check button present during the creation phase. To resolve this issue, it was decided to eliminate the possibility to modify the jira code once it is inserted and checked. In this way, it is avoided the possibility to make errors regarding the effort and plan management of the projects.

This highlights the importance of continuously monitoring and improving the processes to ensure the smooth functioning of the project.

Chapter 6

Conclusions

In conclusion, the development of CoolVision has successfully achieved the objectives set for the creation of the application. The implementation of the solution designed for the management and monitoring of worklogs and staffing has met the specific requirements and needs identified at the beginning of the project. The solution proposed by CoolVision has proved to be highly functional and user-friendly, allowing for a streamlined and efficient management of worklogs and staffing. The integration of various graphs and charts has allowed for easy visualization of data, enabling stakeholders to quickly identify workload imbalances and take corrective action. Compared to the previous spreadsheet-based solution, CoolVision has provided a significant improvement in terms of usability and ease of use. The application's intuitive interface and interactive graphs have made it much easier for users to navigate and understand the data, resulting in a more efficient and effective management of worklogs and staffing. Overall, the development of CoolVision has been a success, providing a much-needed solution for the management and monitoring of worklogs and staffing. The application's user-friendly interface and advanced data visualization features make it an indispensable tool for companies looking to streamline their workflows and improve their productivity. This section will analyze the results both in terms of use of the application and in terms of potential changes that could lead it to improve over time with new features.

6.1 End results and final interviews

The implementation of Coolvision has had a significant impact on the project management process within Coolshop, resulting in several positive outcomes. After interviewing the project managers, it became apparent that one of the most significant benefits of Coolvision has been the **improvement in the flow related to project estimation**. They reported that it is now much simpler to make project

associations with the use of Coolvision. The system's user-friendly interface has made it easier to create new projects and estimate the time and resources needed to complete them. The system can also help identify and eliminate any discrepancies in project estimates, which is particularly important for ensuring projects stay within budget and are completed on time. The project managers also noted that the presence of various graphs and data associated with each project has led to an **improvement in analysis quality**. Coolvision allows project managers to access real-time data, which enables them to track project progress and identify any areas that may require attention. The system also provides the project managers with a better understanding of how projects are progressing compared to their initial estimates. This has allowed the project managers to **adjust their approach to projects**, ensuring they are on track to meet their goals. Another significant benefit of Coolvision is that it has provided project managers with a 360-degree view of employees, allowing them to **make more informed decisions about employee allocation**. The system provides project managers with an overview of the availability of each employee, allowing them to assign tasks to individuals who have the necessary skills and availability to complete them. This has led to more efficient use of employee time, resulting in projects being completed on time and within budget. The project managers interviewed reported an overall **increase in the quality of evaluations** made since the implementation of Coolvision. They found that the system provided them with a better understanding of the projects they were working on, as well as the resources and time needed to complete them. They also noted that the system's user-friendly interface and real-time data made the flow of work more manageable, allowing them to focus on delivering high-quality projects while still meeting their deadlines.

In conclusion, the implementation of Coolvision has had a significant positive impact on the project management process at Coolshop. The system's user-friendly interface, real-time data, and 360-degree view of employees have resulted in improvements in project estimation, analysis quality, and employee allocation. The project managers reported an overall increase in the quality of evaluations made, indicating that Coolvision has struck a balance between usability and functionality, making the flow of work enjoyable and manageable.

6.2 Application usage analysis

The application usage analysis is an important part of any development project. It helps to identify how users are interacting with the application, what features are being used most frequently, and where improvements can be made. After the development of the application, the project managers started using it to manage their projects. The flow of usage revealed several focal points that are worth

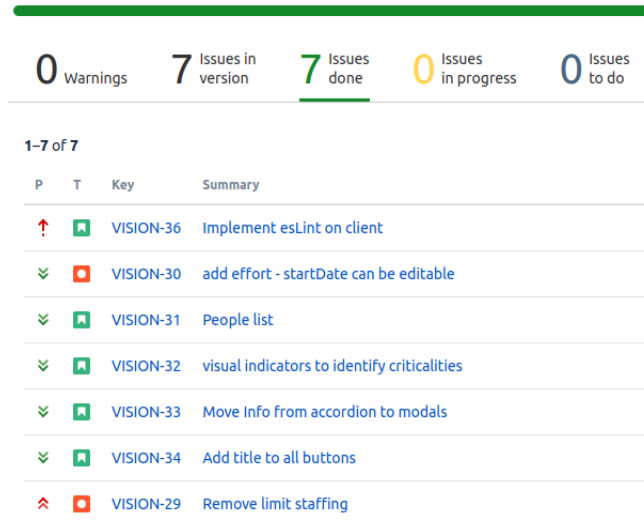
discussing. They are:

1. **Increased Monitoring of Project Statistics:** Once a project is created and effort is added to it, project managers tend to check the statistics related to the hours worked for that particular project more frequently than before. This suggests that the application is providing them with valuable information that they find useful in managing their projects.
2. **Preliminary Analysis of Commercial and Estimate:** In line with the first point, project managers are also more likely to perform a preliminary analysis of commercial and estimate before allocating resources to a new project. They tend to review the work done in the previous period before making any decisions about the allocation of resources.
3. **Increased Focus on Allocation:** There is now more study and analysis involved in the allocation phase than before. Project managers now check the monthly report of projects and the progress of work for the project they want to undertake. Before, allocation was done more instinctively based on customer requests. This suggests that the application is helping project managers to allocate resources more efficiently.
4. **Resource Allocation:** Resource allocation is a fundamental aspect of the application. In most cases, project managers prefer to check the availability of resources for a project in a particular month before adding any effort. This suggests that the application is providing them with a more accurate picture of resource availability, which in turn helps them to manage their projects more effectively.
5. **Quick Comparison between Estimate and Commercial:** The ability to make quick comparisons between the estimate and commercial is having a significant impact on project management. Project managers now associate these two values with wider margins than before, leading to more efficient and consistent evaluations. This suggests that the application is providing project managers with valuable insights into the financial aspects of project management.

The application usage analysis has provided valuable insights into how project managers are using the application to manage their projects. The patterns identified suggest that the application is providing project managers with reliable and useful information that is helping them to make informed decisions about resource allocation, project monitoring, and financial evaluations. These insights can be used to improve the application further and make it an even more effective tool for project management.

6.3 Potential future changes

Based on the analysis carried out in the previous chapters and considering the operational workflow of CoolShop, there are several potential future changes that could improve the application's performance and user experience. The following are the key areas of focus:



The screenshot shows a Jira dashboard with a green header bar. Below the header, there are five summary cards: '0 Warnings', '7 Issues in version', '7 Issues done' (highlighted with a green underline), '0 Issues in progress', and '0 Issues to do'. Below these cards, there is a table of issues. The table has columns for 'P' (Priority), 'T' (Type), 'Key', and 'Summary'. The issues listed are:

P	T	Key	Summary
↑	📌	VISION-36	Implement esLint on client
⌵	📌	VISION-30	add effort - startDate can be editable
⌵	📌	VISION-31	People list
⌵	📌	VISION-32	visual indicators to identify criticalities
⌵	📌	VISION-33	Move info from accordion to modals
⌵	📌	VISION-34	Add title to all buttons
⬆	📌	VISION-29	Remove limit staffing

Figure 6.1: Main features related to the v1.0.1 of Coolvision stored in Coolshop's Jira server

- **Resource Management Section:** A new section within the application entirely dedicated to the company's employees. This section will provide a comprehensive view of which projects are staffed for each reference month, helping to reverse the current paradigm of visualization and focus primarily on resource management. By having a dedicated resource management section, project managers can manage their team's allocation and avoid any potential conflicts or overloading.
- **Code Optimization:** To improve the user experience, the code could be optimized to speed up the calculation of statistics. This change will help to make the application more responsive and fluid, ensuring that project managers can quickly access the information they need.
- **Streamline Monthly Progress Pages:** The pages related to monthly progress can be streamlined by reducing the number of accordions and creating dedicated modals for each project and month. This approach will allow project managers to view all relevant statistics in one place, without having to navigate

multiple screens. Additionally, by incorporating more graphs and visuals, project managers can get a better understanding of the project's progress and make informed decisions.

These key areas of focus have been appropriately scheduled and prioritized for inclusion in the **release 1.0.1** (figure 6.1) of the Coolvision application. By implementing these changes, the application will become more user-friendly, and project managers will be able to manage their resources more effectively.

In conclusion, this thesis has provided an in-depth analysis of the development and usage of the Coolvision application. By exploring the application's features and functionality, as well as its impact on project managers, it has become clear that the application has the potential to revolutionize the way CoolShop manages its projects. The future changes proposed in this chapter will further enhance the application's capabilities and streamline the project management process. Overall, the development of the application has been a significant step forward for the company. The application has demonstrated its value in enabling project managers to efficiently manage their projects, allocate resources, and track progress. The results of the usage analysis have shown that the application is providing value to the company, and there is significant potential for further development. With continued development and refinement, Coolvision has the potential to become an essential tool for project managers in the company and help to ensure that projects are delivered on time and within budget.

Bibliography

- [1] Microsoft Corporation. *Microsoft Project*. [Online; accessed 2023]. URL: <https://www.microsoft.com/en-us/microsoft-365/project/project-management-software> (cit. on p. 1).
- [2] Inc. Asana. *Asana*. [Online; accessed 2023]. URL: <https://asana.com> (cit. on p. 1).
- [3] Atlassian Corporation Plc. *Trello*. [Online; accessed 2023]. URL: <https://trello.com/home> (cit. on p. 1).
- [4] Atlassian Corporation Plc. *Jira Software*. [Online; accessed 2023]. URL: <https://www.atlassian.com/software/jira> (cit. on p. 1).
- [5] Winston W Royce. «Managing the development of large software systems». In: *Proceedings of IEEE WESCON 26* (1970), pp. 1–9 (cit. on p. 2).
- [6] Ken Schwaber. «Agile Project Management with Scrum». In: *Microsoft Press* (2002) (cit. on p. 2).
- [7] David J Anderson. *Kanban: successful evolutionary change for your technology business*. 2010 (cit. on p. 2).
- [8] Kent Beck et al. *Manifesto for Agile Software Development*. Agile Alliance, 2001 (cit. on p. 2).
- [9] Meta and community. *React*. [Online; accessed 2023]. URL: <https://reactjs.org/> (cit. on p. 4).
- [10] Node.js Foundation. *Node.js*. [Online; accessed 2023]. URL: <https://nodejs.org/> (cit. on p. 4).
- [11] GitLab and community. *GitLab*. [Online; accessed 2023]. URL: <https://gitlab.com/> (cit. on p. 4).
- [12] Inc. Docker. *Docker*. [Online; accessed 2023]. URL: <https://docs.docker.com/> (cit. on p. 15).
- [13] Inc. MongoDB. *MongoDB*. [Online; accessed 2023]. URL: <https://www.mongodb.com/> (cit. on p. 16).

- [14] Meta and community. *GraphQL*. [Online; accessed 2023]. URL: <https://graphql.org/> (cit. on p. 16).
- [15] Atlassian. *Jira REST API documentation*. [Online; accessed: 2023]. URL: <https://developer.atlassian.com/cloud/jira/platform/rest/v3/intro/> (cit. on p. 17).
- [16] StrongLoop TJ Holowaychuk et al. *Express.js*. [Online; accessed 2023]. URL: <https://expressjs.com/> (cit. on p. 18).
- [17] Recharts Group. *Recharts: A composable charting library built on React components*. [Online; accessed 2023]. URL: <https://recharts.org/> (cit. on p. 18).
- [18] Material UI SAS. *Material-UI: A popular React UI framework*. [Online; accessed 2023]. URL: <https://material-ui.com/> (cit. on p. 18).
- [19] Valeri Karpov and community. «Mongoose». In: (). [Online; accessed 2023]. URL: <https://mongoosejs.com/> (cit. on p. 19).
- [20] GraphQL Foundation. *express-graphql*. [Online; accessed: 2023]. URL: <https://graphql.org/graphql-js/express-graphql/> (cit. on p. 19).
- [21] Atlassian Corporation Plc. *Jira.js*. [Online; accessed 2023]. URL: <https://mrrefactoring.github.io/jira.js/> (cit. on p. 19).
- [22] Stansio Labs. *The Symfony PHP framework*. [Online; accessed: 2023]. URL: <https://symfony.com/> (cit. on p. 35).
- [23] Doctrine Project. *Doctrine ORM*. [Online; accessed: 2023]. URL: <https://www.doctrine-project.org/> (cit. on p. 37).
- [24] Jared Hanson. *Passport - Simple, unobtrusive authentication for Node.js*. [Online; accessed: 2023]. URL: <http://www.passportjs.org/> (cit. on p. 52).