

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO



Infrastructure Management Tool

Francisco Fernandes Machado

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Supervisor: Prof. Hélio Mendonça

June 28, 2021

Resumo

A presente dissertação, realizada em parceria com a Bosch Car Multimédia Portugal S.A., teve como objetivo o desenvolvimento de uma solução para a gestão do inventário da secção XC-12/ESD5. Os requisitos e objetivos desta ferramenta já estavam definidos pelo cliente - secção XC-12/ESD5.

Numa primeira instância, foi realizada uma análise de mercado relativa a ferramentas existentes cujo a funcionalidade era semelhante ao proposto nesta dissertação. Procurou-se perceber as principais vantagens e desvantagens de cada ferramenta, bem como estudar funcionalidades que poderiam acrescentar valor à solução desenvolvida. Em seguida, tendo em conta que nenhuma ferramenta encontrada satisfazia os objetivos do cliente, e era necessário desenvolver uma solução de raiz, foi feita uma revisão literária relativa à arquitetura de uma aplicação *web* e de possíveis tecnologias a ser implementadas.

Como esta dissertação foi realizada em ambiente empresarial, foi imprescindível seguir a metodologia de trabalho da empresa. Para rapidamente estar familiarizado com esta metodologia, foi importante perceber os conceitos de Agile e de Scrum.

Posteriormente, o desenvolvimento da solução foi iniciado. Após uma breve discussão dos requisitos, o sistema que viria a ser implementado começou a ser desenhado com a construção de diversos diagramas UML - Unified Modeling Language, *mockup* das páginas e ainda os *endpoints* da API - Application Programming Interface.

Depois da arquitetura do sistema estar definida, a ferramenta começou a ser desenvolvida de forma autónoma. Durante o desenvolvimento, foram ainda utilizadas *frameworks* de teste para garantir a integridade da solução.

No final, o trabalho realizado foi avaliado tendo em conta a realização dos objetivos propostos e ainda as funcionalidades existentes. Também foram propostas algumas melhorias futuras.

Abstract

This thesis, elaborated in partnership with Bosch Car Multimédia Portugal S.A., aimed to develop a solution for managing the inventory of the XC-12/ESD5 section. The requirements and objectives of this tool were already established by the client - section XC-12/ESD5.

First of all, a market research regarding existing tools whose goal was similar to the intended in this dissertation was carried out. The main objective was to comprehend the primary benefits and drawbacks of each tool, as well as examine features that could add value to the developed solution. Then, taking into account that none of the tools found satisfied the client's objectives, and it was necessary to develop a solution from scratch, a literature review regarding web applications architecture and technologies was accomplished.

Since this dissertation was performed in a business environment, it was essential to follow the company's work methodology. To quickly become familiar with this methodology, it was important to understand the concepts of Agile and Scrum.

Later, the development of the solution started. After a brief discussion of the requirements with the client, the system that would be implemented began to be designed with the construction of several UML diagrams, page mockups, and API endpoints.

After the system architecture was defined, the tool began to be developed. During the development phase, two test frameworks were used to ensure the integrity of the solution.

In the end, the work done was evaluated through the achievement of the proposed objectives and the existing features. Some future improvements were also proposed.

Acknowledgments

First of all, I would like to thank Bosch Car Multimédia Portugal, S.A., for the trust and opportunity to write the dissertation in a company like Bosch and for the provided conditions to accomplish my work.

I would also like to especially thank my enterprise supervisor, Engenheira Ana Rita Martins, for her availability and guidance throughout the project. Thank you for always having time for my questions, it was undoubtedly crucial for the success of the project.

I would like to thank my FEUP coordinator, Professor Hélio Mendonça, for all the support provided along the pathway and for accepting to be my thesis supervisor.

Por fim, quero agradecer a todos os meus amigos e família por toda a paciência e apoio, não só neste período, mas também nos últimos cinco anos. Muito obrigado a todos, a vossa confiança e carinho motiva-me a ser uma pessoa melhor.

Francisco Fernandes Machado

“A sorte dá muito trabalho.”

Contents

1	Introduction	1
1.1	The Bosch Group	1
1.1.1	Bosch Group History	1
1.1.2	Bosch Car Multimédia Portugal, S.A.	2
1.2	Project Framework and Motivations	3
1.3	Main Goals	3
1.4	Thesis Outline	4
2	Literature Review	5
2.1	IT Asset Management	5
2.1.1	ServiceNow Asset Management	6
2.1.2	EZOfficeInventory	7
2.1.3	ManageEngine AssetExplorer	10
2.1.4	Lansweeper	11
2.1.5	Asset Panda	13
2.1.6	Snipe-IT	15
2.1.7	Spiceworks	17
2.2	Web App Development	18
2.2.1	Frontend	18
2.2.2	Backend	30
3	Work Methodology	39
3.1	Agile Values and Principles	39
3.2	Scrum Framework	40
3.3	Kanban Method	44
3.4	Project Methodology	46
3.4.1	RTC	46
4	System Overview and Design	49
4.1	System Requirements	49
4.2	Class Diagram	56
4.3	Use Case Diagrams	58
4.3.1	Basic	58
4.3.2	Scrum Master	60
4.3.3	Admin	61
4.4	Sequence Diagrams	63
4.4.1	Signup	63
4.4.2	Login	63

4.4.3	Edit User Information	64
4.4.4	Add Component	65
4.4.5	Edit Component	66
4.4.6	Delete Component	67
4.4.7	Add Target	68
4.4.8	Edit Target	69
4.4.9	Delete Target	70
4.4.10	Add Cables	71
4.4.11	Edit Cables	72
4.4.12	Delete Cables	73
4.4.13	Acknowledge Notification	74
4.4.14	Clear Notification	75
4.4.15	Acknowledge All Notifications	76
4.4.16	Overtime Notification	77
4.5	User Interface Mockup and Structure	77
4.5.1	/login	79
4.5.2	/signup	79
4.5.3	/main	80
4.6	Backend API Endpoints	81
5	System Implementation	85
5.1	System Setup	85
5.2	Third-party Libraries Integration	86
5.2.1	React Router DOM	86
5.2.2	React Hook Form	87
5.2.3	Material-UI	87
5.2.4	Express	87
5.2.5	Mongoose	88
5.2.6	Bcryptjs	88
5.2.7	Jsonwebtoken	88
5.2.8	Web Push	88
5.3	Main Features	89
5.3.1	Authentication and Authorization	89
5.3.2	Data Validation	90
5.3.3	User Management	91
5.3.4	Components and Targets Management	95
5.3.5	Cables Management	99
5.3.6	History	102
5.3.7	Search and Catalog	105
5.3.8	Notification System	106
6	Testing and Validation	115
6.1	API Testing	115
6.2	End to End Testing	116
7	Conclusion	119
7.1	Summary	119
7.2	Objective Fulfillment	119
7.3	Future Work	120

A	Appendix	123
A.1	Prices	123
A.2	UI Mockup	123
A.3	Extra Application UI	124
	References	135

List of Figures

1.1	Bosch Logo	1
1.2	Bosch Car Multimédia Portugal Headquarters	2
2.1	Lansweeper ITAM 2.0	11
2.2	Multi-Page Application Lifecycle vs SPA Lifecycle	20
2.3	Angular Architecture	22
2.4	Vue Ecosystem	25
2.5	Number of npm packages downloaded	26
2.6	Percentage of Stack Overflow questions each year	27
2.7	Perfomance when creating or appending rows in a table (Duration in milliseconds)	28
2.8	Memory usage when creating or appending rows in a table (Memory in MB)	29
2.9	REST Architectural Style	31
2.10	GraphQL	33
2.11	REST API vs GraphQL	35
3.1	Scrum Team	41
3.2	Scrum Events whitin a Sprint	43
3.3	Kanban Board	44
3.4	RTC User Story Example	47
3.5	RTC Sprint Kanban Board	47
4.1	System Class Diagram	58
4.2	Basic User Use Case Diagram	60
4.3	Scrum Master Use Case Diagram	61
4.4	Admin Use Case Diagram	62
4.5	Signup Sequence Diagram	63
4.6	Login Sequence Diagram	64
4.7	Edit User Information Sequence Diagram	64
4.8	Add Component Sequence Diagram	65
4.9	Edit Component Sequence Diagram	66
4.10	Delete Component Sequence Diagram	67
4.11	Add Target Sequence Diagram	68
4.12	Edit Target Sequence Diagram	69
4.13	Delete Target Sequence Diagram	70
4.14	Add Cables Sequence Diagram	71
4.15	Edit Cables Sequence Diagram	72
4.16	Delete Cables Sequence Diagram	73
4.17	Acknowledge Notification Sequence Diagram	74
4.18	Clear Notification Sequence Diagram	75

4.19	Acknowledge All Notifications Sequence Diagram	76
4.20	Overtime Notification Sequence Diagram	77
4.21	Application Pages	78
4.22	Login UI Mockup	79
4.23	Sign up UI Mockup	80
4.24	Main UI Mockup	81
4.25	API Endpoints	83
5.1	MERN Stack	85
5.2	Deployment Diagram	86
5.3	Authentication	90
5.4	Login UI	91
5.5	Login Sequence Diagram	92
5.6	Signup UI	93
5.7	Signup Sequence Diagram	93
5.8	Edit User Information UI	94
5.9	Edit User Information Sequence Diagram	95
5.10	Example Catalog UI	96
5.11	Add Component Sequence Diagram	97
5.12	Edit Component Sequence Diagram	98
5.13	Delete Component Sequence Diagram	99
5.14	Add Cables Sequence Diagram	100
5.15	Edit Cables Sequence Diagram	101
5.16	Delete Cables Sequence Diagram	102
5.17	Component History UI	103
5.18	Components History Sequence Diagram	103
5.19	User History UI	104
5.20	User Components History Sequence Diagram	104
5.21	Extended View UI	105
5.22	Components Catalog Sequence Diagram	106
5.23	Notifications Menu UI	108
5.24	Notifications Menu Sequence Diagram	109
5.25	Acknowledge Notification Sequence Diagram	110
5.26	Acknowledge Over Time Notification Sequence Diagram	111
5.27	Clear Notification Sequence Diagram	112
5.28	Acknowledge All Notifications Sequence Diagram	113
6.1	Postman Collections	115
6.2	Postman Request	116
6.3	Cypress HTML report	117
A.1	EZOfficeInventory Plans and Prices with 250 items	123
A.2	EZOfficeInventory Plans and Prices with 14750 items	124
A.3	ManageEngine AssetExplorer Subscription Model	125
A.4	Snipe-IT Pricing	125
A.5	Components Catalog UI Mockup	126
A.6	Targets Catalog UI Mockup	127
A.7	Cables Catalog UI Mockup	128
A.8	User Information UI Mockup	129

A.9 Notifications Menu UI Mockup 130

A.10 Add Component UI 130

A.11 Edit Component UI 131

A.12 Success Message UI 131

A.13 Delete Message UI 131

A.14 Confirmation Dialog UI 132

A.15 Error Modal UI 133

List of Tables

2.1	Tools	6
7.1	Goals and Achievement Degree	120
7.2	Infrastructure Management Tool Features	120

Abbreviations

API	Application Programming Interface
AWS	Amazon Web Services
CSS	Cascading Style Sheets
DBMS	Database Management System
Dev	Developer
DOM	Document Object Model
ECDSA	Elliptic Curve Digital Signature Algorithm
E2E	End to End
GPS	Global Positioning System
HMAC	Hash-based Message Authentication Code
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IP	Internet Protocol
ITAM	Information Technology Asset Management
IoT	Internet of Things
JSON	JavaScript Object Notation
JSX	JavaScript XML
JWT	JSON Web Token
LDAP	Lightweight Directory Access Protocol
MAC	Media Access Control
MERN	MongoDB, Express, React, Node
MPA	Multiple Page Application
MVC	Model–View–Controller
ODM	Object Data Modeling
PO	Product Owner
QR	Quick Response
REST	Representational State Transfer
RFID	Radio Frequency Identification
RSA	Rivest-Shamir-Adleman
RTC	Rational Team Concert
SAML	Security Assertion Markup Language
SDK	Software Development Kit
SQL	Structured Query Language
SPA	Single Page Application
SSL	Secure Socket Layer
TLS	Transport Layer Security
UI	User Interface
UML	Unified Modeling Language
URL	Uniform Resource Locator
US	User Stories
VAPID	Voluntary Application Server Identity
VPC	Virtual Private Cloud

Chapter 1

Introduction

In this first chapter, a brief description of Bosch Group including Bosch Global and Bosch Car Multimédia Portugal, S.A. information will be presented in section 1.1. Afterwards, in section 1.2, the project framework and motivations are described. In section 1.3 and section 1.4 the goals and the work methodology will be introduced, respectively. Finally, in section 1.5 the thesis outline is detailed.

1.1 The Bosch Group

1.1.1 Bosch Group History

Robert Bosch GmbH commonly known as Bosch, Figure 1.1, was founded by Robert Bosch in 1896 in Stuttgart, Germany. The company started as a workshop for precision mechanics and electrical engineering.

Nowadays, the Bosch Group is a "leading global supplier of technology and services". It employs roughly 394 500 associates worldwide. According to preliminary figures, the company generated sales of 71.6 billion EUR in 2020. Mobility Solutions, Industrial Technology, Consumer Goods, and Energy and Building Technology are the four business sectors in which it operates.



Figure 1.1: Bosch Logo

Bosch, as a leading IoT vendor, offers cutting-edge smart home, Industry 4.0, and linked mobility solutions. Bosch is working toward a vision of mobility that is environmentally friendly, safe, and fun. It offers connected, cross-domain solutions from a single source, leveraging its experience in sensor technology, software, and services, as well as its own IoT cloud. The strategic goal

of the Bosch Group is to make connected living easier through products and solutions that either contain artificial intelligence or were created or manufactured using it.

Bosch improves people's lives throughout the world by providing innovative products and services that drive excitement. In short, Bosch creates technology that is "Invented for life"[1].

1.1.2 Bosch Car Multimédia Portugal, S.A.

The history of Bosch Car Multimédia Portugal in Braga dates back to 1990, when the Blaupunkt facility first opened for business. The plant was dedicated to the production of vehicle radios at the time, and it held a significant position in the region and country.

With the changing demands of the automobile market, the brand was sold in 2009, and the firm was reorganized to become Bosch Car Multimédia Portugal, SA (Figure 1.2) which is committed to the research and production of infotainment, instrumentation, and sensors systems.



Figure 1.2: Bosch Car Multimédia Portugal Headquarters

Nowadays, this unit focuses on the development and production of multimedia solutions and automotive sensors. Bosch Car Multimédia Portugal also hosts teams from other divisions in the Mobility area, such as Cross-Domain Computing Solutions, Chassis Systems and Automotive Aftermarket. At Bosch in Braga, we also find several areas of the Group's central services such as CI - Information Systems and Services and GS - Global Services. The company exports almost all of its output, placing it among the top exporting Portuguese enterprises in the country. It is also one of the largest employers in the Braga region[2].

1.2 Project Framework and Motivations

This master thesis project is developed in collaboration with the XC-12/ESD5 section at Bosch Car Multimédia Portugal, S.A. This section is responsible for the development of software that is integrated into the two-wheel clusters which are designed by the other sections of the department.

The project proposed is the development of a tool capable of managing the inventory of this section, instead of using an Excel spreadsheet. This tool must meet the requirements defined by the Bosch section, which may change, and meet the final objectives, described in section 1.3. For the development of the platform, the XC-12/ESD5 section gave complete freedom in choosing the technologies and architecture of the solution.

Even though this project seems out of the scope of what is developed in this department, the truth is that it's a much-needed tool. For the engineers to assemble or test the clusters, they must have the physical components. The amount of existing hardware, in use or available, for each project and each section, is vast. To manage these goods it is necessary to keep on record the material availability, who's currently owning it, if none, where is it, which section the equipment belongs to, the material cost, type, serialID, etc. It is also necessary to improve how this information is managed: not everyone can change it, what happens if someone forgets to update the component status or edits it wrongfully. In order to fulfill these needs, using a spreadsheet is not enough, it is necessary to have an infrastructure management tool capable of handling these data and improve the user interaction and experience. This application also minimizes the mistakes inadvertently made by the user: the data modifications are way more under control than, for example with Excel.

The truth is that there are already many tools available for inventory management, however, many are expensive, do not offer the chance for customize, or cannot be integrated with Bosch's existing services. This is why the decision was to develop a tool from scratch: to solve the client problem, working side by side with the section, and, at the same time, have a customizable platform to match the client's needs and easily integrated into the existing systems.

1.3 Main Goals

Like the high-level requirements and desired features, the main goals of this project were already defined by the client (XC-12/ESD5 section at Bosch Car Multimédia Portugal):

- Have a user-friendly tool for the team.
- Have the inventory stored in a server and accessible to all team members.
- Manage the information and share it with the user when requested.

1.4 Thesis Outline

The remaining parts of this thesis are structured into the following chapters:

- **Chapter 2 - Literature Review:** In this chapter, a market research regarding IT Asset Management tools is pursued and necessary concepts for understanding web application development are reviewed.
- **Chapter 3 - Work Methodology:** This chapter, provides a general idea of what is Agile and how the work was structured and planned within the student and the company in an agile way.
- **Chapter 4 - System Overview and Design:** This chapter describes the system requirements and architecture of the proposed solution.
- **Chapter 5 - System Implementation:** This chapter provides implementation details of the proposed solution.
- **Chapter 6 - Testing and Validation:** In this chapter, the methods used to test and validate the application solution are presented and exemplified.
- **Chapter 7 - Conclusion and Future Work:** In this chapter, the summary of the work is presented. In addition, proposals for future development are discussed.

Chapter 2

Literature Review

In this chapter, the concept of IT Asset Management is explained as well as a market research for existing products is carried out. Consequently, the fundamentals and technologies of web application development are described.

2.1 IT Asset Management

Information Technology Asset Management - IT Asset Management or ITAM - " is a combination of tools and processes that proactively manage a company's entire asset base from a cost, contractual, support, and inventory viewpoint" [3]. Assets take an important role in an organization since they are used by employees and contractors to create value or improve profits within a business corporation. Thus, "companies must possess certain competencies to maximize their effectiveness in each area(assets, people, and profits)". Those competencies are:

- **Deploying IT assets:** Implementation, maintenance, inventory, and report of IT assets. Nowadays, companies use auto-discovery tools that find every hardware component and installed software within a network. This competence supports the assets area but it does not "provide the type of data useful for understanding information about a company's regular business processes".
- **Help desks:** This competence is related to the people part. "To get the maximum value out of its IT assets", an organization must provide continuous support to ensure that the employees can work efficiently and consequently maximize company profits. To achieve this goal "companies have made sizable investments in software, network control centers, and personnel to staff help desks and reduce user downtime".
- **Managing ownership data:** The first two competencies regard the asset's current state - "these tools do not tell you about the asset as it was in the past or as it should be." However, that data "is necessary for determining ownership information" - purchase orders, contractual terms and conditions, assets history, etc. Companies need that kind of knowledge to

make accurate decisions about assets. That will lead "to lowered expenses and increased revenues".

A well-implemented ITAM can avoid unnecessary asset purchases by redeploying assets rather than buying new ones, for example, optimize software licenses use while reducing associated costs, manage maintenance contracts, make the best use of the current resources, eliminate waste, improve productivity through technology support, provides a single source of truth for IT assets and increases the company business value while reducing the expenses.

Nowadays, there are already numerous tools that offer an IT Asset Management service. In Table 2.1, several solutions are analyzed taking into consideration the project required features and the price:

Table 2.1: Tools

Tool	User Authentication	User Management	Roles	Asset Information	Asset History	Notification System	Catalog	Bosch Integration	Price
ServiceNow Asset Management	Yes	Yes	Yes	Yes	Yes	No	Yes	No	Paid
EZOfficeInventory	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Paid
ManageEngine AssetExplorer	No	No	No	Yes	Yes	Yes	Yes	No	Paid
Lansweeper	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Paid
Asset Panda	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Paid
Snipe-IT	Yes	No	No	Yes	Yes	Yes	Yes	Yes/No	Free/Paid
Spiceworks	Yes	Yes	No	Yes	Yes	No	Yes	No	Free

In the next subsections, we will look with detail into these solutions and understand better what are the provided functionalities, and why those tools don't fit the project goal.

2.1.1 ServiceNow Asset Management

The ServiceNow Asset Management is a ServiceNow solution that manages the IT assets - "it tracks the financial, contractual and inventory details of hardware and devices – as well as non-IT assets – throughout their lifecycle" [4].

This solution is part of various IT workflows which are all unified in a single platform named Now Platform, a cloud-based platform where "anyone can digitize and automate departmental and cross-enterprise workflows, optimize business processes for resiliency, and mitigate risk...".

The key features of this tool are [5]:

- **Asset Provisioning:** When there is a new asset request, the tool can quickly check if there is available inventory and, if there are no items in stock, "Asset Management makes it easy to create a purchase order with a single click".

- **Service Catalog:** Asset Managers can list items into a catalog service and define which groups of users are authorized to request the asset.
- **Inventory Management:** Customers can "manage stockroom inventory of hardware and consumable assets and define physical and logical stockroom hierarchies."
- **Track Total Cost of Ownership:** It's possible to track every cost associated with an asset through its lifecycle.
- **Mobile Asset Receiving:** "Receive multiple assets against a purchase order in a single process on the mobile agent app" in order to reduce manual data entry and risk of error.
- **Mobile My Assets:** The user can view and request assets as well as report incidents related to his/her assets through a mobile device.
- **Contract Management:** Contract management capabilities where the process of renewal and contract approval is automatic.

The main benefits of this tool to achieve the final objective - ITAM - is that: it is easier to know and control the assets, optimize the assets lifecycle reducing costs and resources, as well as limiting risks. Thus, we can say that this solution fulfills all the needs of this project, yet that's not entirely true.

Even though this tool has relevant functionalities, some are missing, such as a notification system, and there are others not mentioned above that are not part of the scope of this project, HR asset on-boarding, for example. That's what makes this solution a bit overwhelming for this work: the ServiceNow Asset Management has features that do not belong to the main goals of this project and the platform is aligned with the full management of an enterprise and its operations.

This solution is perfect for a business organization that intends to manage all of its assets, which is not the case. Bosch already has their platform to manage most of their assets, there is no sense in migrating to a whole new service to solve a specific problem of a minor part of the organization.

2.1.2 EZOfficeInventory

EZOfficeInventory is a web-based software that is capable of tracking and managing assets through their lifecycle, simplify operational workflows and manage the inventory, tag items with asset labels, and customize user control [6].

The key functionalities of this tool can be split into four different features:

1. Asset Management

- **Asset Lifecycle Management:** "End-to-end asset lifecycle management from procurement to retirement. Add and track asset purchase orders, vendors, maintenance histories, and lifetime costs."
- **Asset Stock Inventory Module:** Track the asset stock and inventory.
- **Check out Check-in Assets:** Check assets in and out with ease using their equipment management software.
- **Mobile Apps:** It's possible to scan Barcodes and QR - Quick Response - Codes from the equipment and access their information at any time, anywhere.
- **Availability Calendar:** The status of the assets is shown: available, reserved, checked out, or under maintenance.
- **Reservations:** "The Reservations module enables you to plan ahead and ensure conflict-free bookings."
- **Purchase Orders:** Keep track of the assets that belong to Purchase Orders and manage costs.
- **RFID Asset Tracking and GPS Location**
- **Label Scanner and Designer:** "Create professional-grade labels of different sizes, styles, and formats using the label designer"
- **Groups, Subgroups, Packages, and Bundles:** Classify assets based on characteristics and usage. The assets can be treated as a single entity or separately.
- **Audits and Calibration Management:** Define regular checkpoints through the asset lifecycle by setting up, for example, custom alerts.

2. User Management

- **Custom Roles:** Specify what actions can the user take or which information can the user access.
- **Arbitration:** Increase control by requiring administrator approval before employees can reserve and check items out from the system.
- **Login Enabled:** The users need to log in to the system to use it. The access can be made through Google/Outlook.com credentials or the organization's LDAP - Lightweight Directory Access Protocol - credentials.

3. Maintenance Management

- **Service Tickets:** "Create tickets to schedule and keep track of maintenance events. "
- **Recurring Services and Scheduled Maintenance:** Schedule recurring maintenance or plan it in advance.
- **Maintenance Alerts:** "Send out alerts to relevant parties on service initiation, completion, extension, or comments".

- **Service History:** Show a snapshot of all services related to a specific asset.

4. Reports Notifications

- **Custom Alerts:** Send customized emails for specific events.
- **Custom Reports:** "Extensive productivity-boosting reports".
- **Data Backups:** Data exports to Dropbox automatically if desired.
- **Scheduled Reports:** "Schedule custom reports to be sent to company users at regular intervals to eliminate time spent gathering, compiling, and sharing these data."

This tool offers a wide variety of features that match the needs of the project. There's also an API feature - "Integrate EZOfficeInventory with other systems using our REST based APIs", that would make the integration with the Bosch system easier, yet we need to take into consideration all aspects of this product. What about the price? Is it free? Does it depend on the desired features or the number of items? This information is available in the Figure [A.1](#) and [A.2](#).

It's important to define what counts as items: it's a "sum of all uniquely identifiable Assets, Asset Stock and Inventory. Individual stock quantities do not count towards the package limit". For example:

- **Assets:** "All active assets count towards the package limit. This does not include retired assets".
- **Asset Stock:** "All Asset Stock items count towards the package limit but each item's stock quantity is not taken into account. For example, if you want to track 500 chairs and 200 tables, your total Asset Stock count is 2, not 700".
- **Inventory:** "All Inventory items count towards the package limit but each item's stock quantity is not taken into account. For example, if you want to track 100 juice boxes, 50 pens, and 50 notepads, your Inventory count is 3, not 200".

The number of items can be above 14 750 but it is the maximum amount where there is a price already defined for each subscription package. The number of users follows a fair usage policy where the number of users in the system can be up to twice the number of items.

As is visible in the figures mentioned, there are three types of subscription packages: Essential, Advanced, and Premium. Each plan has more features than the previous one but the price is also higher. It's the same logic for the number of items, if the quantity increases the costs grow. It is fair to say that this solution is scalable but so is the price. This tool can work as an independent platform or could be integrated with the Bosch network. To combine this solution with the Bosch services, it would be necessary to use the API integration, and for that, the chosen package should be the Advanced subscription, at least.

In summary, the EZOfficeInventory has a lot of interesting powerful features but it is also expensive. Once again, if we were looking to manage all assets of an organization and make use of all the features provided by this solution, it would be a perfect fit yet that's not the case.

2.1.3 ManageEngine AssetExplorer

ManageEngine AssetExplorer is a web-based software that "helps you monitor and manage assets in your network from Planning phase to Disposal phase" [7]. The AssetExplorer manages software and hardware assets through their lifecycles, including Purchase Orders and Contracts tracking, the total cost of ownership, software license compliance, and informed decisions about new purchases.

The seven main features of this tool are:

- **Configuration Management Database (CMDB):** It's a centralized repository that stores information on all the existing significant entities, known as Configurations Items (CIs). It also establishes and maintains relationships between CIs.
- **IT Asset Inventory Management:** All the assets are in the network categorized and ready to be efficiently monitored or managed. The assets can be added using various methods: windows domain scans, network, and network range scans, individual asset scans, manual import, or through the mobile app.
- **Asset Tracking Software:** Hardware and software inventory tracking, associated vendors and contracts, managing asset groups, assets relationship, and history.
- **Software Asset Management:** "Includes software discovery, software license compliance, software forecasting, and ensuring optimum usage of the bought licenses".
- **Contract Management:** Manage the contracts with vendors, create contracts and associate with any asset, expiry dates or renewal alerts, and the possibility to attach the term and conditions related to a contract.
- **Purchase Order Management:** This system manages the cycle of a purchase order: creation, approval, contact the vendor, receive the assets, and process the invoice details and the payment.
- **Mobile App:** The user can add, manage and track all the existing assets anytime, anywhere with the ManageEngine AssetExplorer mobile app.

The Figure [A.3](#) shows the license fee annual price for various IT assets. There is no limit to the number of users.

This solution, taking into account the number of features and asking price is not worth it. Reports and notifications, user management, the total cost of the ownership, API integration are examples of functionalities that this tool doesn't provide. Lastly, the price of 14 750 assets using the Advanced package, which includes the features mentioned above, from the EZOfficeInventory is 9090 USD annually and the cost of 10 000 assets using ManageEngine AssetExplorer is 11 195USD, without those features. We can conclude that this solution is not a great choice because we are getting less and paying more when compared with other products in the market.

2.1.4 Lansweeper

Lansweeper is a cross-platform software that can discover, analyze, control, and coordinate an entire IT network. They also propose a new version of the traditional IT Asset Management - the ITAM 2.0 - a scenario-independent approach [8].

Nowadays, when "new hardware and software assets are introduced into an organization's IT infrastructure, it becomes increasingly difficult to streamline your IT Asset Management. Managing various IT scenarios" - for each new scenario is created a new inventory and each tool collects the needed data for their use case - "has resulted in the creation of silos, making it difficult to maintain a complete, universal view."

The ITAM 2.0, described in the Figure 2.1, is a scenario-independent approach, instead of collecting data for a specific IT scenario, it collects all the information. Thus, "organizations can create an always-accurate single source of truth to inform all IT scenarios and strategic decision-making across the entire organization". To achieve this goal the solutions divides into four key areas: Asset Discovery, IT Inventory, IT Analytics, and Integrations.

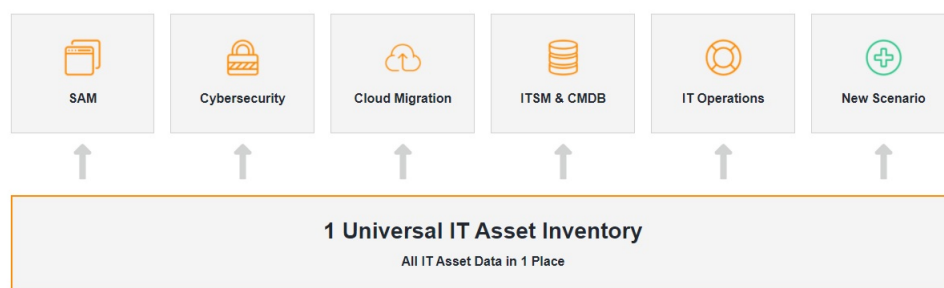


Figure 2.1: Lansweeper ITAM 2.0
[8]

The Asset Discovery is responsible for finding all the existing IT assets that are connected to a network - Windows, Linux, and Mac devices, printers, routers, and switches - or assets that are not yet deployed or not connected to the network.

- **Lansweeper Deepscan:** Discovers any asset that is connected to the network without having to install any software on the machines.
- **Asset Radar:** Detects assets the moment they connect to the network and also identifies and recognizes the assets thanks to an AI technology - Credential-free Device Recognition.
- **Active Network Discovery:** Agentless IP Range Scanning, Agentless Active Directory Scanning, Agentless Workgroup Scanning, Exchange Office 365 Scanning, Windows Cluster Scanning, Cloud Server Discovery, Chrome OS Discovery, SCCM Discovery, Mobile Device Discovery, and Agent-Based Scanning with LsAgent.

The IT Inventory automatically collects complete hardware, software, and user information from all IT assets and builds a centralized inventory.

- **Hardware Inventory:** Collects information about computers and their hardware components, and builds a hardware inventory from scratch.
- **Software Inventory:** "Automatically scans software on Windows, Linux, and Mac devices, and gets a complete software inventory and license audit of your IT environment."
- **User Inventory:** Complete overview of all the users and the assets that they are related to.

The IT Analytics provides complete visibility and insights from the IT environment [8].

- **Network Reports:** "Reports gather specific information based on criteria in a convenient overview". It is possible to create customized reports.
- **IT Dashboard:** "Convenient and customized overview of relevant information about your IT environment"
- **Software License Compliance:** Tracks software license keys, license purchases, product pricing and creates detailed reports where you can find all this information.
- **Vulnerability Updates:** Delivers the latest vulnerabilities allowing the user to detect a network vulnerability before it's exploited.
- **Patch Tuesday Report:** Checks if the IT Assets are running on the latest Microsoft Patch Tuesday Update.

Lansweeper offers the possibility to integrate their solution to an existing ecosystem: 4me, Corent SurPaaS, ServiceNow Service Graph Connector, CyberStockRoom, ServiceNow CI Synchronizer, HalolTSM, Splunk, and License Manager.

This tool also has other interesting features that add value to the solution:

- **Custom Actions:** Enables the user to remotely perform tasks on one specific device. These actions can be either built-in actions or customized.
- **Email Alerts:** Automatically send an email to a specific email address or group when a report or event log alert is performed.
- **Asset Location Tracking:** Know where the assets are located.
- **Asset Uptime Calendar:** "The uptime calendar tracks when a device was turned on, in sleep mode, and turned off".
- **Automated Software Deployment:** "Create deployment packages, set conditions to check before deploying, and execute commands to remotely apply changes to Windows computers in your network"
- **In-App Notifications:** Lansweeper has integrated notifications that can be customized or personalized.

Lansweeper has two types of subscription packages: Free and Enterprise. The first one as the name says it's free but up to 100 assets. The second one costs 1 EUR per asset, paid annually. Both plans include all the features and unlimited users. An asset is any device or resource scanned by Lansweeper - "Linux, Windows, Mac computer, VMware server, any other network device (printer, switch, firewall, etc.), virtual machines, resource group, VPC (Virtual Private Cloud), etc".

This solution offers a variety of features that match the needs of this master thesis project. Nevertheless, it's an expensive tool and it's more focused on the management of assets that are connected to a network and that's not the case of most of the assets that will be managed.

2.1.5 Asset Panda

Asset Panda is a high customizable cloud-based technology that manages and tracks assets. It allows the user to configure the platform as he/she wants with "unlimited fields and configurations", to define and assign permissions, and to "have on-demand access to asset information anytime, from anywhere". The features of Asset Panda are grouped into five sections [9]:

1. Asset Tracking

- **Asset assignment:** Assign assets to users.
- **Asset kitting:** Group assets to perform actions across multiple types of equipment efficiently.
- **Barcode generation:** Design and print custom barcodes or QR codes.

- **Check-out and returns:** "Custom actions to deploy and return assets and keep everyone in the loop with past-due reminders and up-to-date data."
- **Compliance tracking:** Ensure that the compliance guidelines are followed.
- **Contract management:** Notifications are sent whenever an asset is set to expire to ease renewals or renegotiations.
- **Full lifecycle history:** Access to the history of any asset.
- **Maintenance tracking:** Scheduled maintenance notifications.
- **Repair tracking:** "Simplify repair requests to get the equipment back up and running quickly".
- **Software license tracking:** Keep a record of software licenses assigned to devices and/or users.
- **Record-level attachments:** Attach any file to an equipment record for "faster troubleshooting and easier identification".

2. Configuration

- **Action-based forms:** Custom check-out forms and maintenance checklists.
- **Asset listing view:** Assets catalog with the chance to apply filters or grouping.
- **Client-defined workflow actions:** "Pre-built actions like check-in/out, repair, transfer, and dispose to quickly update your asset records or define your own actions to match your workflows".
- **Custom fields:** Define custom data and details.
- **Custom groups:** Define custom groups "like assets, employees, buildings, and more".
- **Custom reports:** Asset history reports are available in various formats and can be automatically sent.
- **Field requirements:** Ensure data integrity with required fields.
- **Notifications:** Email or push notification "when maintenance is due, actions are performed, or fields meet your defined criteria".

3. Mobile app

- **iOS and Android Compatible.**
- **Full integration with web app:** "Use a robust web app with full administrative features or access and update your data on the go."
- **Built-in barcode scanning:** Scan barcodes or QR codes to view or update asset information.
- **Reporting access:** Downloadable reports.

- **Quick asset addition:** "Take a photo and fill in details on your smartphone to rapidly add assets into your account wherever you are."

4. Ease of use

- **User-specific views:** Display a view according to the user's role.
- **Sortable columns.**
- **Built-in search:** "Enter text or scan a barcode to rapidly access asset records".
- **Data import:** Import existing data on your assets.
- **Easy record replication:** Instantly copy record details into new assets.
- **Unlimited users.**
- **Bulk asset reallocation:** "Save time by updating multiple fields across a group of assets all at once".

5. Security

- **Role-based user security:** Limit user access based on user role.
- **Amazon AWS hosting:** "Cloud-based EC2 hosting with S3 storage, backed by Amazon's global network security procedures".
- **256-bit SSL encryption:** "Protection for data exchanges at the highest available level of SSL - Secure Socket Layer - encryption".

Asset Panda also offers integration with two Microsoft services: Microsoft Active Directory and Microsoft System Center Configuration Manager. That's relevant because it can simplify the integration with the Bosch services since they also use Microsoft technologies to manage their network administration and security.

Asset Panda does not fully disclose its pricing structure, and the only information accessible is that for 1500 USD annually, they support 500 assets and all the available features [9].

To conclude, this solution has everything. It offers essential features, integration services, and has security but still has one drawback, it's paid.

2.1.6 Snipe-IT

Snipe-IT is an open-source asset management cloud-hosted platform powered by AWS - Amazon Web Services. It focuses on the user experience, security, frequent updates, dedicated support, and custom integrations achieved with a powerful REST - Representational State Transfer - API [10].

The main features of this product are:

1. General Features:

- Web-based software
- "Mobile-friendly for asset updates on the go"
- Slack notification integration
- LDAP login/user sync
- SAML - Security Assertion Markup Language - login integration

2. Asset and License Management:

- See which assets or licenses are assigned, to whom, and where they are
- One-click check-in
- Asset Models share common features
- User Acceptance on Checkout
- Integrates with most barcode scanners and QR code reader apps
- Asset auditing
- Customize asset attributes
- Import and export assets
- Generate QR code labels
- Request system
- Access to history of an asset (checkouts, check-ins, maintenance)

3. Admin Dashboard:

- Recent activity - "Items that have been checked in, checked out, recently updated or deleted"
- Overall view of all assets, licenses, accessories, and consumables

4. Email Notifications:

- Asset Check-out/Check-in sent to Users and Admins
- Check-in Deadline sent to Users
- Expected Check-in Report, Expiring License Report, and Low Inventory Report sent to Admins

5. JSON - JavaScript Object Notation - REST API:

- Easy integration with other systems
- Well-documented API

- New open-source SDK - Software Development Kits - are being developed by the community

6. Security:

- "One-way secure password hashing with bcrypt"
- "Encrypted fields secured via AES-256 encryption via OpenSSL"
- "SQL - Structured Query Language - injection prevention using prepared statements"
- "All connections secured via TLS - Transport Layer Security - (SSL)"
- "Snapshots and individual data backups tested regularly"

Even though Snipe-IT is a free tool they offer paid hosting packages, visible in the Figure A.4, where they "take care of the server set-up, the installation, all of the maintenance and upgrades, plus priority support for any issues that might come up". The only way to use this solution completely free is to self-host the platform. The functionalities that are set aside when we choose to self-host Snipe-IT are the support ones: API integration help, priority feature requests, email support, phone support, and their platform benefits: automatic backups, automatic upgrades, server maintenance, and TLS/SSL encryption.

To conclude, Snipe-IT offers several features that match the project requirements and is a free service. The only problem this solution may have is the integration with Bosch technologies which can lead to the choice of a paid package - either for professional/enterprise support or their hosting, which is not the desired option.

2.1.7 Spiceworks

Spiceworks is a free online IT Management software that can manage users and the network. The tool splits into five different use-cases: Help Desk, Connectivity Dashboard, Cloud Help Desk, and Inventory [11].

The Inventory tool is responsible for managing IT assets that are connected to a network. The key features of this solution are:

- **Asset Management System:** Scans all the hardware with an IP - Internet Protocol - address that is connected to the network. It also gathers related information such as: "manufacturer, MAC - Media Access Control - address, open ports, hardware details, and installed software."
- **Software and Licenses Tracking:** Automatically scans all of the existing software and tracks and audits the software licenses.
- **Manage Tickets for Devices:** Track all device's tasks and information.

- **Custom Reporting:** Built-in reports or custom reports on all network assets.
- **Scheduled Scans:** Automated inventory scans: "Set the time and/or frequency you need to make sure everything is up-to-date and not slowing down your network at inopportune times"s
- **Fully Customizable:** Full-featured plugins to reports and language packs in the Spiceworks Community that can be used to customize the Spiceworks app.
- **Clients and Users Tracking:** Track all users and clients and all related "devices, tickets, purchases, contact info, and even photos."

Spiceworks also includes completely free support and unlimited assets, tickets, agents, and users. They also offer the client the choice to self-host or to host online, both options are free.

To summarize, the main advantage of this tool is the fact that is totally free. On the other hand, it lacks a lot of features and the possibility to integrate with the existing Bosch technologies.

2.2 Web App Development

As the solution for this project is the development of a tool from scratch without any framework restriction, it's necessary to understand how it can be built and which technologies can be used.

Web app development is the process of creating software that is hosted on remote servers and that can be accessed by the user through a web browser with an active network connection. Web application development is typically subdivided into three phases:

- **Front-End:** Client-side programming.
- **Back-End:** Server-side programming.
- **Database:** Used to store data.

In the next sections, these concepts will be explained in more detail as well as the respective technologies.

2.2.1 Frontend

The client-side of a web application is everything that the user can see and interact with, and it's referred to as the frontend. The front end is synonymous with the user interface from the user's perspective. The interface design and code are what make the interface work from a developer's perspective [12, 13].

The creation of a smooth or frictionless user experience is one of the main goals of frontend development. In other words, an application's front end should be user-friendly. While this may appear to be a simple goal, it can be surprisingly hard to accomplish. A mobile app, for example, requires a very different frontend than a desktop application - websites must work efficiently across a variety of devices and screen sizes, which is why responsive design is so important in current web development.

The three main elements required for building the user interface are:

- **HTML:** Hypertext Markup Language (HTML) is the language responsible for defining the web page structure. Over the past few decades, HTML got simpler and now relies on CSS - Cascading Style Sheets - styling or JavaScript to format nearly all the elements within a page.
- **CSS:** Cascading Style Sheets (CSS) is the language responsible for styling the web page: it describes how HTML elements are displayed.
- **JavaScript:** JavaScript is a "client-side scripting language, which means the source code is processed by the client's web browser rather than on the webserver". It is responsible for adding logic and dynamicity to the frontend.

2.2.1.1 Single Page Applications

A single-page application (SPA) is a web application that interacts with the user by dynamically rewriting the current page rather than loading an entire new page from a server. This results in a more comfortable experience for the user, and one that is not continually interrupted with successive page navigations. A SPA approach "allows the user to continue consuming and interacting with the page while new elements are being updated or fetched, and can result in much faster interactions and content reloading" [14, 15] .

Each page of a typical multi-page application is written on the server as HTML and sent to the client browser, as visible in the Figure 2.2. Each page has its URL - Uniform Resource Locator, which the client uses to request. The whole page loads when a user navigates from one page to another. However, many UI - User Interface - components are identical across all pages, including the sidebar, header, footer, navigation menu, login/logout UI, etc. As a result, downloading these similar parts on each page request is wasteful. Moving from one page to another may be inconvenient in terms of user experience.

There is just one URL in SPA. The app and all of its necessary components are loaded in the first client request. These files could be HTML plus JavaScript/CSS. This initial bundle of files could be substantial if the application is complex. As a result, the app's first view may take some time to load. A loading graphic may be displayed on the screen at this phase.

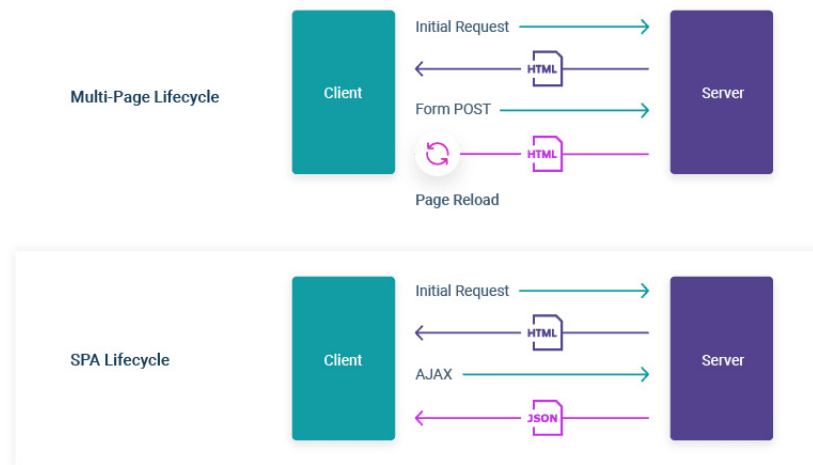


Figure 2.2: Multi-Page Application Lifecycle vs SPA Lifecycle [16]

When you click a link, it downloads relevant content and updates specific UI components to render that content. The user experience improves since the user remains on the current page and may interact with it while the new content is retrieved from the server through an API. There is no transition to another page when an update occurs. The only thing that changes is the view, which is updated by altering some of the page's components [17].

Single-page applications main advantages are [18, 19, 20]:

- **Performance and Speed:** The majority of the resources that a SPA requires - HTML, CSS, and scripts - are loaded when the app is first launched and do not need to be reloaded throughout use.
- **User Experience:** Rich UI content with a simple and modern user experience is given by SPAs, which are straightforward to port between platforms. It provides a seamless user experience, thanks to the fact that only some components are rendered when the user changes the "page".
- **Debugging:** Using the network tab and inspect tools, SPA makes it simple to troubleshoot application code.
- **Caching:** The application just sends one request, saves all transmitted data, and can use it. In SPA applications, difficulties such as session management using HTTP - Hypertext Transfer Protocol - session methods and transferring data between pages are not encountered.

SPA also have their disadvantages which are [18, 21]:

- **Search Engine Optimization:** Search engine optimization is based on HTML content, the fewer pages with unique URL, headers, and keywords a site has, the lower it will rank.

- **Website stats:** Another factor to consider is the impact of a single-page application on the website's statistics. Given that there is only one page, it's impossible to know which page or content is the most popular. Google Analytics, for example, keeps track of data from different pages that are downloaded on a website. It is necessary to build an analytics system if there is a wish to track the activity on a SPA.
- **Security:** Hackers can obtain access to your website and inject new scripts on the client-side thanks to cross-site scripting and the fact that no new pages are loaded.
- **JavaScript:** First of all, the client must enable JavaScript in his browser, otherwise the application doesn't work. When there is a lot of JavaScript data on a page, the download speed can be slow, which impairs the user experience. There can also exist memory leaks because it's the same page that is always running.

Single Page Applications are "well-suited when we wish to build rich interactive UI with lots of client-side behaviour" [17]. Since the main goal of this project is to have a user-friendly tool, and one way of achieving it is by having an interactive user interface, SPA is the number one choice. It is also a great option if there is the need to develop a mobile app in the future, since, as mentioned above, the backend API can be used for both web and mobile apps. The main drawback of an SPA, the search engine optimization, doesn't apply here considering this project is restricted to Bosch use, so there's no need to have optimization search on Google, for example.

2.2.1.2 Angular,Vue,React

Angular, React and Vue are the main frameworks for building the frontend of web applications, especially single-page applications. In this section, these frameworks will be described and compared into consideration: popularity, ecosystem, performance, and learning curve.

Angular is a full-fledged model-view-controller and TypeScript-based open-source framework run by Google. It gives clear instructions on how to organize the application and supports bi-directional data flow with real DOM - Document Object Model. It also includes [22]:

- A framework for creating scalable web applications, mainly single-page applications, based on components.
- A set of well-integrated libraries that cover a wide range of functions, such as routing, form management, client-server communication, and more.
- A bundle of developer tools that helps in the development, building, testing, and updating of the software.

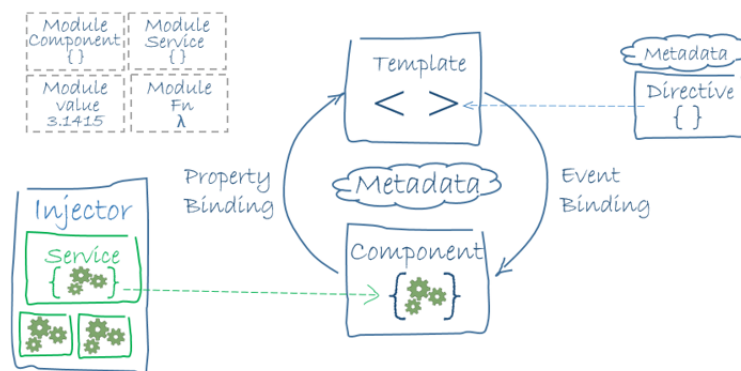


Figure 2.3: Angular Architecture
[23]

The architecture of an Angular application, Figure 2.3, relies on certain fundamental concepts. The basic building blocks of the Angular framework are [23, 24, 25, 26]:

1. **Modules:** The Angular application is modular. NgModules is the modularity mechanism built into Angular. Each Angular application includes at least one Angular module class.
2. **Components:** Components are the pieces that build up an application. A TypeScript class with a Component decorator, an HTML template, and styles make up a component.
3. **Templates:** The angular template mixes HTML with Angular syntax, which allows the edition of HTML components before they're displayed.
4. **Metadata:** Angular uses metadata to determine how to handle a class.
5. **Data Binding:** In the communication between a template and its component, data binding is crucial. There are two types of data binding:
 - Event binding: The application responds to user input in the target environment by updating the application data.
 - Property binding: Lets the user interpolate values that are computed from the application data into the HTML
6. **Directives:** Angular's templates are dynamic by default. Attribute and structural directives are the two types of directives available in Angular. Structural directives can easily adjust layouts by replacing, adding, or removing items in the DOM. Attribute directives are used to change the behavior or appearance of a current element.
7. **Services:** A service class is developed when you have data or logic that isn't related to the view but needs to be shared across components. Attribute directives are used to change the behavior or appearance of a current element.

8. **Dependency Injection:** Keeps your component classes lean and efficient. They don't fetch data from the server, validate user input, or log directly to the console; they delegate such tasks to services.

React is a JavaScript library for creating interactive and fast user interfaces for web and mobile apps. It's a component-based, open-source front-end library that's just responsible for the application's view layer. The view layer is in charge of how the program feels and looks in the model view controller architecture. It was created by Jordan Walke, a software engineer at Facebook, in 2013 [27, 28].

Components are at the foundation of all React apps. A component is a self-contained module that produces a particular result. As a React component, it is possible to develop interface elements such as a button or an input field. Components can be put together in many ways. In its output, a component may incorporate one or more additional components. To look at it another way, it is possible to build React components that correlate to various UI elements to create React apps. The components are subsequently organized within higher-level components, which constitute the application's structure. There are two types of components in React [29]:

- **Functional Components:** These components are also known as stateless components because they have no state of their own and just have a render method. They may derive data from other components as props.
- **Class Components:** Also known as stateful components, because these components can hold and manage their state. They also have a separate render method for returning JSX - JavaScript XML - on the screen.

The state is a built-in React object that holds data or information about the component. The state of a component can change over time, and when it does, the component re-renders. The state of the component might change in response to user activity or system-generated events, and these changes impact how the component behaves and renders.

Props is a built-in React object that saves the value of tag attributes and functions similarly to HTML attributes. It allows users to send data between components in the same manner that arguments are passed between functions.

In summary, components, state, and props are React's fundamental concepts. React also includes several features such as [29, 28, 30]:

- **JSX:** JSX is a JavaScript syntactic extension. It's a term that's used in React to describe how the user interface should look.

- **Virtual DOM:** The virtual DOM - Document Object Model - is a stripped-down version of the original DOM. It has all of the characteristics of a true DOM. When a change is made, it affects the entire virtual DOM. This may seem inefficient, yet virtual DOM is significantly faster than the original. React compares the updated virtual DOM to the pre-updated original DOM after the update is complete. React knows exactly where the difference is after comparing both DOMs. As a result, it only updates the original DOM where there is a difference.
- **One-way data-binding:** One-way data binding indicates that data only flows in one direction throughout the application. This allows you to have more control over it.
- **React Native:** React is the foundation of React Native, providing all of the benefits of React. Because React Native and React are so similar, it allows for a smooth transfer from web applications to mobile apps.

Vue is a framework for creating progressive user interfaces. Vue is built from the ground up to be incrementally adoptable, unlike other monolithic frameworks. The core library focuses solely on the view layer, and is simple to use and combine with other libraries or projects. Vue, on the other hand, when combined with contemporary technology and accompanying libraries, is perfectly capable of powering powerful Single-Page Applications. It was created by Evan You when he was working at Google on Angular-related projects. He combined the best aspects of existing libraries, including Angular's two-way data binding and directives, React's virtual DOM implementation, and Angular's templating syntax [31, 32].

Vue has a rich ecosystem, shown in the Figure 2.4, "of official core libraries, great developer tooling support, a galaxy of third-party plugins, and developer tutorials":

- **Vue Router:** Vue official router library.
- **Vuex:** Vue library for state management. It acts as a centralized repository for all of an application's components, with rules ensuring that the state can only be changed in predictable ways.
- **Vue Loader:** This allows the user to make Vue components in a format called Single-File Components.
- **Vue Server Renderer:** Vue-server-renderer makes it easier to create isomorphic or universal JavaScript apps that operate on both the server and the client-side and share the majority of the code.
- **Vue Test Utils:** Vue official unit testing library.
- **Vue Dev-Tools:** Browser dev tool extension for debugging Vue applications.
- **Vue CLI:** Vue user-interface component libraries.

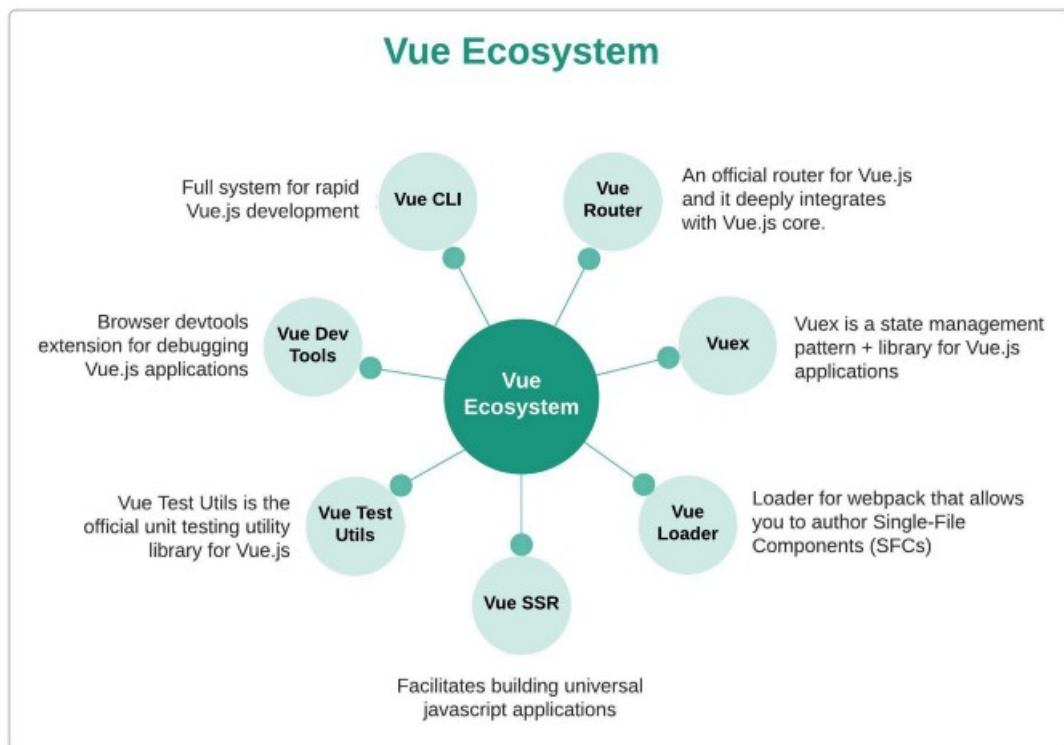


Figure 2.4: Vue Ecosystem
[31]

In summary, Vue is, like Angular, a full-fledged framework.

The first factor that will be compared the three frameworks is their popularity. To measure this parameter we will look into different sources of information and statistics. In the Figure 2.5, the number of npm packages downloaded is presented:

According to the npm trends website, which measures the number of packages downloaded over time, React is the leading framework with 10 000 000 million downloads, five times the number of downloads of Angular or Vue [33].

Stack Overflow Trends compute how technologies have trended over time based on the use of their tags since 2008 when Stack Overflow was founded. React, once again, leads the charts as visible in Figure 2.6. It's the tag that is more common when asking a question in Stack Overflow when compared to Vue or Angular. It shows great community support and interest by the users [34].

The last parameter that will be taken into consideration is the number of GitHub stars and repositories each framework has:

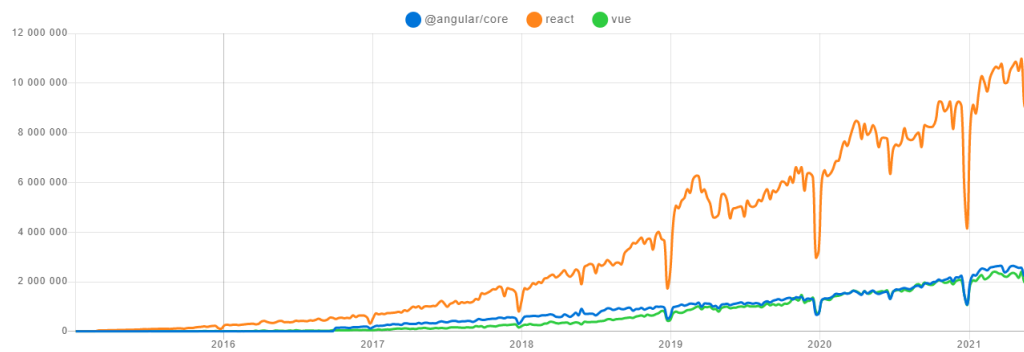


Figure 2.5: Number of npm packages downloaded [33]

- **Angular:** 73 734 stars and 1 843 216 repositories [35].
- **React:** 169 478 stars and 6 647 258 repositories [36].
- **Vue:** 184 230 starts and 2 079 644 repositories [37].

In the end, React is by far the most popular framework, except for the GitHub stars. It is the library that is most used and most searched in the community. This ensures an immense flow of data when looking for answers or technical information, easing the development and smoothing the learning curve.

The second characteristic that will be taken into consideration when comparing these three frameworks, is the environment. It is critical to consider the availability of ready-to-use components, global state management, themes, and other tools that can assist in the development of new features [38, 39, 40].

- The NgRx project can be used for state management in Angular. There are many ready-to-use components import into projects, but the Angular Material project has several official components, which makes it slightly distinct from other projects. This is a Google-sponsored project that provides Material Design components for Angular apps. Angular has its own testing tool - Karma. NativeScript allows you to create cross-platform mobile apps in Angular. Angular is part of the MEAN stack that combines Angular with MongoDB, ExpressJS, and NodeJS. It relies entirely on JavaScript for both the front-end and back-end.
- Redux is the most popular JavaScript state management project and it's used together with React. Thanks to React popularity, it's easy to discover ready-to-use components - Material UI for example. React Testing Library is a set of helpers that let you test React components without relying on their implementation details. React Native is a component of the React ecosystem that allows you to create native iOS and Android apps using a single React codebase. As a result, React might be an alternative for developing mobile apps using web

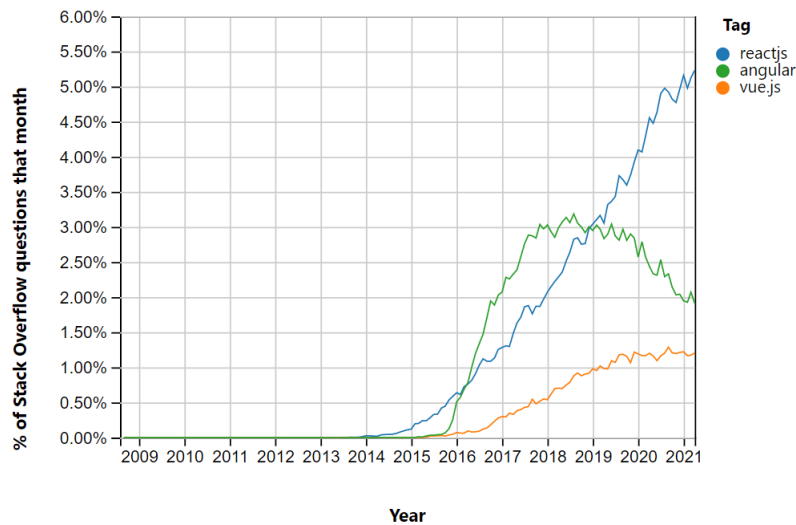


Figure 2.6: Percentage of Stack Overflow questions each year
[34]

technologies. React is part of the MERN stack that combines React instead of Angular with MongoDB, ExpressJS, and NodeJS.

- As stated before, Vue has its own state management, testing unit, and user interface component libraries - Vuex, Vue Test Utils, and Vue CLI respectively. For mobile app development, Vue Native but it's not as mature and powerful as React Native. For the backend, Vue goes well with Laravel.

Overall, all frameworks have a solid environment: state management is granted either by third-party libraries or the framework itself, ready-to-use components are easy to find and implement, mobile app development is also possible and all have a development stack.

Performance is another factor that must be accounted for when choosing a front-end framework or library. The more a project scales and grows in complexity, performance can (and will) become a problem. It's necessary to measure this metric and evaluate which framework brings out the best result [41, 42]. In the next figures, the green color means good performance, yellow satisfying performance and red poor performance.

When highlighting a selected row, Vue has the worst performance when compared to the others frameworks, as shown in Figure 2.7. In reverse, React and Angular have similar but worst performances when compared to Vue when swapping rows.

Memory usage is similar in all frameworks for the tests presented. Vue has the best performance as can be seen from the geometric mean and shown in the Figure 2.7.

Name Duration for...	vue-v3.0.6	angular- v12.0.1	react- v17.0.1
Implementation notes			
create rows creating 1,000 rows	106.3 ± 2.6 (1.00)	135.7 ± 1.5 (1.28)	138.7 ± 1.3 (1.30)
replace all rows updating all 1,000 rows (5 warmup runs).	93.2 ± 1.2 (1.00)	109.0 ± 1.6 (1.17)	112.7 ± 1.5 (1.21)
partial update updating every 10th row for 1,000 rows (3 warmup runs). 16x CPU slowdown.	153.4 ± 8.5 (1.08)	145.1 ± 1.8 (1.00)	193.2 ± 2.2 (1.33)
select row highlighting a selected row. (no warmup runs). 16x CPU slowdown.	147.4 ± 3.0 (2.05)	71.9 ± 1.8 (1.00)	102.6 ± 3.5 (1.43)
swap rows swap 2 rows for table with 1,000 rows. (5 warmup runs). 4x CPU slowdown.	45.0 ± 1.0 (1.00)	349.0 ± 1.9 (7.76)	342.6 ± 1.7 (7.62)
remove row removing one row. (5 warmup runs).	21.3 ± 0.4 (1.14)	18.7 ± 0.3 (1.00)	21.8 ± 0.9 (1.17)
create many rows creating 10,000 rows	922.6 ± 4.1 (1.00)	1,093.9 ± 15.5 (1.19)	1,470.9 ± 38.7 (1.59)
append rows to large table appending 1,000 to a table of 10,000 rows. 2x CPU slowdown.	221.3 ± 4.0 (1.00)	251.0 ± 4.3 (1.13)	263.2 ± 2.0 (1.19)
clear rows clearing a table with 1,000 rows. 8x CPU slowdown.	106.4 ± 0.8 (1.00)	215.7 ± 5.2 (2.03)	125.0 ± 1.5 (1.17)
geometric mean of all factors in the table	1.11	1.47	1.57

Figure 2.7: Performance when creating or appending rows in a table (Duration in milliseconds) [42]

In summary, performance is a big factor and must be taken into real consideration. Vue takes the best performance in both figures but the overall results are similar. It's vital to remember that when it comes to web performance, the quality of programming and following best practices are more important than the chosen framework.

Last but not the least, the framework learning curve. The learning curve refers to a user's ability to write code in a particular programming language and to how difficult it is to learn that language [41, 43].

Angular is the most complicated project structure of the three, and it relies on more concepts because it's a full-fledged front-end framework. Because of its usage of TypeScript, Angular has the steepest learning curve of the three frameworks. Although learning Angular takes time, the investment pays off in terms of understanding how the front end operates.

The React learning curve in comparison to Angular and Vue, js has a medium to a high learning curve. React utilizes JSX syntax, which is a hybrid of JavaScript and HTML that many users find confusing because it looks like HTML and acts like JavaScript. This is the key factor that makes React's learning curve a little steeper.

Name	vue-v3.0.6	react-v17.0.1	angular-v12.0.1
ready memory Memory usage after page load.	1.2 (1.00)	1.3 (1.00)	1.8 (1.48)
run memory Memory usage after adding 1000 rows.	3.5 (1.00)	4.4 (1.25)	4.2 (1.19)
update eatch 10th row for 1k rows (5 cycles) Memory usage after clicking update every 10th row 5 times	3.7 (1.00)	5.2 (1.42)	4.5 (1.23)
replace 1k rows (5 cycles) Memory usage after clicking create 1000 rows 5 times	4.1 (1.00)	5.2 (1.28)	5.2 (1.28)
creating/clearing 1k rows (5 cycles) Memory usage after creating and clearing 1000 rows 5 times	2.6 (1.00)	3.0 (1.15)	3.5 (1.38)
geometric mean of all factors in the table	1.00	1.23	1.31

Figure 2.8: Memory usage when creating or appending rows in a table (Memory in MB) [42]

Vue has the easiest learning curve when compared to the others. The simplicity of the syntax and the fact that is very similar to HTML makes the learning curve straightforward.

React is the chosen library for this project. After careful and thoughtful analysis, React was chosen as the framework to be used for the development of the web application frontend. The factors that had the most weight were: ease of learning, the fact that it is more popular and, in terms of performance, it is at a similar level to the others. The learning curve is less steep than the others and it has a huge impact on this project because the more time it takes to learn how to work with the framework more the start of the project development is delayed. The fact that it is popular means that you can easily find a lot of information on the internet, answers to problems, and community support.

The choice of the remaining stack follows as a consequence of the choice of React for the frontend. The MERN stack, MongoDB for database, Node.js and Express for the backend server, and React for frontend, was chosen because of the easy integration of these frameworks, easing the learning and the integration of solutions.

Also in this section, the use of JavaScript or TypeScript for the project programming language is discussed.

In the next section, instead of reviewing the existing technologies for the backend, the architecture, the database concept, and a brief description of MongoDB is presented.

2.2.1.3 JavaScript and TypeScript

JavaScript is a high-level scripting language that helps to create interactive and dynamic web pages. JavaScript is also recognized by its dynamic type and just-in-time compiler. It also supports functional programming, imperative programming styles, and event-driven programming, making it a multi-paradigm language. When the script executes in the user's browser, JavaScript uses a client-side implementation. It does, however, have engines that support server-side implementations [44, 45, 46].

TypeScript is a superset of JavaScript which was created to handle and develop larger applications by being tightly typed and including compile-time error controls. TypeScript supports static typing and can follow objected oriented programming principles. It can use existing JavaScript code, incorporate popular JavaScript libraries, and can be called from other JavaScript code.

As seen so far TypeScript is a JavaScript superset. This makes the languages seem similar, which is not consistent with reality. What are the key differences between JavaScript and TypeScript?

- **Compilation:** The TypeScript code needs to be compiled while there is no need to compile JavaScript. TypeScript has a compile-time error feature that, compiles the code and checks for compilation errors that may be found in the syntax. In JavaScript, since there is no compiler, it needs to run first to be tested and considered valid or not. Consequently, it takes a long time to find bugs and errors in the code.
- **Language:** JavaScript is a scripting language, whereas TypeScript is next to an object-oriented programming language.
- **Typing:** TypeScript provides static typing while JavaScript doesn't. It enables the developer to verify type accuracy during the compilation process.
- **Learning Curve and Community:** Typescript does not have a large developer community and the learning curve is steep. On the other hand, JavaScript has a large development community and it's easy to learn.

JavaScript was chosen because the time needed to learn and master TypeScript would be significant and impact negatively the rest of the work. TypeScript is also better for larger projects and JavaScript language was already known by the user.

2.2.2 Backend

The back-end is the code that runs on the server and receives requests from clients, as well as the logic that determines what data to send back to users. The database, which will be used to store all of the data for the application, is also part of the back-end.

The clients are anything that sends queries to the backend. Browsers frequently make requests for HTML and JavaScript code, which they will utilize to show webpages to end-users.

The server is a computer that is listening to the incoming requests made by the client. A server can be any computer that is connected to a network. The server runs an app that contains the logic necessary to understand the client requests, retrieve the information from the database and send the response to the user [47].

An API - Application Program Interface - "is a group of protocols and definitions used to integrate different components of a system." [48]. It is a contract between an information provider and an information user, establishing the content required from the consumer (the call) and the content required by the producer (the response).

A server-side API is a programmatic interface to a defined request-response message system that consists of one or more publicly exposed endpoints. It is characterized by the types of requests it can handle, which are decided by the routes it defines, as well as the types of answers clients can expect after hitting those routes.

The following subsections will focus exclusively on the discussion between a REST API and GraphQL since the backend languages are already defined. Database concepts are described in section 2.2.2.2.

2.2.2.1 REST API and GraphQL

A REST API is an architectural style, Figure 2.9, for an application program interface that uses HTTP requests to access and use data. It is based on REST - Representational State Transfer - a set of architectural constraints. For an API to be considered RESTful it must meet the requirements defined by Dr. Fielding in his 2000 doctorate dissertation [49, 50, 51, 52]:

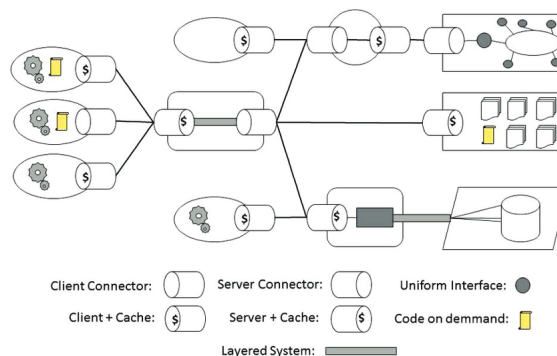


Figure 2.9: REST Architectural Style

[53]

- **Client-Server:** The client-server restrictions are based on the idea of separation of concerns. In order to improve the portability of the user interface across many platforms and improve the scalability of the server, it's necessary to split the user interface concerns from the data storage concerns.
- **Stateless:** REST API are stateless. Each client-to-server request must include all of the information required to understand the request and cannot rely on any server-stored context. As a result, the session state is preserved exclusively on the client. The characteristics of visibility, reliability, and scalability are all imposed by this constraint.
- **Cache:** One of the disadvantages of the stateless REST API is the fact that data cannot be left on the server in a shared context, so it may reduce network performance by raising the repetitive data (per-interaction overhead) transmitted in a series of requests. To solve this issue, the cache must be added to the RESTful API architecture.
- **Uniform Interface:** The uniform interface between components is what distinguishes the REST architectural style from other network-based designs. It allows the application to evolve independently of the API layer, without the services, models, or actions of the application being tightly tied to the API layer.
- **Layered System:** The layered system permits an architecture to be made of hierarchical layers by restricting component behavior such that each component cannot "see" beyond the current layer with which they are interacting.
- **Code-On-Demand:** It is an optional REST constrain. REST allows for code or applets to be transmitted via the API for use within the application.

The idea of REST is made up of these constraints. Looking back, it's possible to see how each restriction builds on the last, culminating in a complex, robust and versatile API. REST APIs also have their challenges regarding security, request and data, endpoints, API versioning, and API testing. These difficulties will be approached later in this subsection when comparing REST APIs to GraphQL.

GraphQL, Figure 2.10, is both an API query language and a runtime for executing those queries using current data. GraphQL allows clients the power to ask for exactly what they need and nothing more, makes it easier to evolve the API over time, and enables powerful developer tools by providing a clear and comprehensible description of the data in the API. It was created by Facebook in 2012 to reduce its mobile applications network usage through specific data-fetching capabilities. GraphQL main design principles are [54, 55, 56]:

- **Hierarchical:** The majority of modern product development focuses on the building and manipulation of view hierarchies. A GraphQL query is hierarchically structured to match

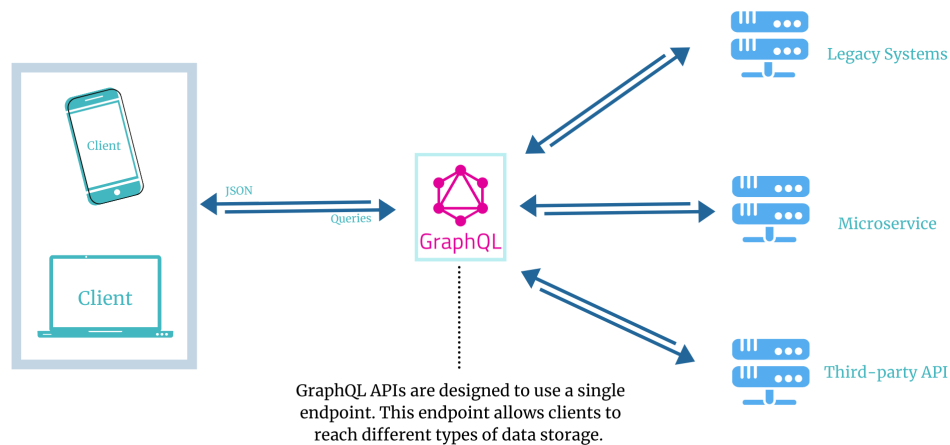


Figure 2.10: GraphQL
[53]

the structure of these applications. The query has the same structure as the data it returns. It's a perfect way for users to express their data needs.

- **Product-centric:** The requirements of views and the frontend engineers who design them, are driving GraphQL. GraphQL takes their reasoning and requirements as its starting point and constructs the language and runtime to support them.
- **Strong-typing:** Every GraphQL server has its own type of system for applications. Queries are run inside the context of the type system in question. Before executing a query, tools can check that it is both syntactically accurate and complete within the GraphQL type system.
- **Client-specified queries:** In most of the client-server applications written without GraphQL, the server determines the data returned in its various endpoints. GraphQL, on the other hand, returns exactly what the client wants nothing more. It is the client that is responsible for specifying how the capabilities announced by the GraphQL server are used.
- **Introspective:** Introspection with GraphQL is a great way to create common tools and client software libraries. The GraphQL type system must queryable by the GraphQL language.

"The goal of this specification is to provide a foundation and framework for an ecosystem of GraphQL tools, client libraries, and server implementations – spanning both organizations and platforms – that has yet to be built."

Now that the fundamental principles and requirements of these concepts are known, the differences between REST API and GraphQL are, Figure 2.11 [56, 53, 57, 58, 59]:

- **Endpoints:** There are several endpoints in a REST API, that retrieve resources by accessing different pathways for different types of data. On the other hand, GraphQL has just one endpoint where the queries or mutations are used to define requests.
- **Data-Fetching:** GraphQL eliminates data over-fetching and under-fetching, which occurs when an endpoint responds to a call with too much or too little data in comparison to what the app requires. REST answers are known for providing either too much or too little data, creating the need for one or more requests. GraphQL, on the other hand, just has one endpoint through which we can access data on a server. It retrieves the exact data with a single request.
- **Error Handling:** In a REST API, error handling is simple. We simply scan the HTTP headers to determine the status of a response. When using GraphQL via HTTP, however, it will always get a 200 OK response status. When an error occurs while processing GraphQL queries, it's necessary to check the whole response that is sent to the client and look for the error message.
- **Caching:** When using a RESTful API the client can use HTTP caching to avoid having to reload resources. In GraphQL there isn't a built-in caching mechanism and it doesn't rely on HTTP caching methods.
- **Learning Curve:** GraphQL takes a while to understand when compared to REST. It is necessary to learn the Schema Definition Language, for example, while REST is much more straightforward and intuitive.
- **Versioning:** When using a third-party REST API is usual to see v1,v2 v3,etc. regarding the different versions of the API. This leads to code redundancy and less maintainable code. In GraphQL, there is no need for versioning since it's easy to add new fields to the schema or mark the old ones as deprecated.

To conclude, both REST API and GraphQL have their pros and cons. GraphQL has just one endpoint, no versioning and avoids over or under data-fetching. REST has a better error handling, HTTP caching and a less steeper learning curve.

In this project the REST architecture will be implemented. The time to learn and understand the GraphQL ecosystem can delay the timing of the project. In the future, if there is time left, GraphQL can be implemented as well - both frameworks can happily co-exist in the same application - since it adds value to the project and the developer.

2.2.2.2 Database

A database can be defined as a shared collection of interrelated data designed to meet the varied information needs of an organization. A database is generally managed by a database management

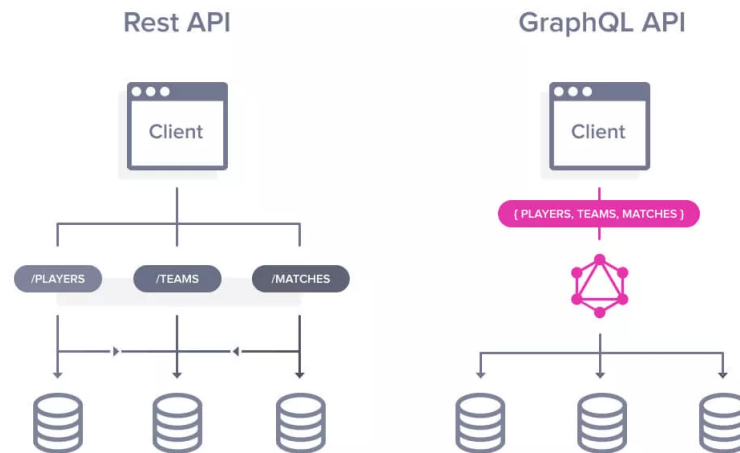


Figure 2.11: REST API vs GraphQL
[60]

system - DBMS. The data that is typically stored in a database, is modeled in rows and columns in a series of tables to make processing and data querying efficient, and can then be easily accessed, managed, modified, updated, controlled, and organized [61].

A database has three principal characteristics:

- **Integrated:** Different data files were previously logically organized to eliminate or reduce redundancy and to make data access easier.
- **Shared:** All eligible users in the organization have access to the same data, which they can utilize for a variety of purposes.
- **Interrelated:** Structured in a logically meaningful way for the organization.

Databases come in a variety of shapes and sizes. Each model has its own set of processing capabilities as well as business benefits. The most known type of databases are [62, 63, 64]:

- **Relational databases:** The data in these databases is organized into predefined categories using a series of tables. The table is made up of rows and columns, with each column containing data for a given category and each row containing an instance of that data determined by the category. The Structured Query Language - SQL - is a relational database's standard user and application software interface.
- **Non-relational databases:** Non-relational database, NoSQL, in contrast, allows unstructured and semistructured data to be stored and manipulated. They provide flexible schemas and scale easily with large amounts of data and high user loads.

- **Object-oriented databases:** An object-oriented database is a combination of relational databases with object-oriented programming. Objects, rather than actions, and data, rather than logic, are the building blocks of an object-oriented database.
- **Distributed databases:** A distributed database is made up of two or more files that are stored at various locations. These sites are linked together through communication links, making it easier to access the spreaded data.
- **Graph databases:** The graph is made up of nodes and edges, representing an entity and a relationship between nodes. A graph-oriented database, often known as a graph database, is a type of NoSQL database that stores, maps, and queries relationships using graph theory.
- **Cloud databases:** A cloud database is a structured or unstructured collection of data stored on a private, public, or hybrid cloud computing platform. A cloud database has several advantages, including the option to pay for storage capacity and bandwidth on a per-user basis, as well as scalability on demand and high availability.

As stated in section 2.2.1, the database that will be used in this project is MongoDB. Next up, MongoDB is detailed.

2.2.2.3 MongoDB

MongoDB is a document-oriented NoSQL database for storing large amounts of data. MongoDB uses collections and documents instead of tables and rows, as in traditional relational databases. Documents are made up of key-value pairs, which are MongoDB's basic data unit. Collections are the equivalent of relational database tables in that they include sets of documents and functions [65].

MongoDB has several features such as:

- **Ad-hoc queries for optimized, real-time analytics:** MongoDB is the cloud database platform of choice for enterprise applications that demand real-time analytics for this reason. The performance boost from ad-hoc query capabilities, which allows developers to update ad-hoc queries in real-time, can be game-changing.
- **Indexing appropriately for better query executions:** MongoDB provides a wide range of indices and features, including language-specific sort orders, to accommodate complex dataset access patterns.
- **Replication for better data availability and stability:** When your data is stored in a single database, it is vulnerable to a variety of failures, including server crashes, service outages, and even hardware failure. By establishing additional servers for disaster recovery and backup, replication allows the user to avoid these risks.

- **Sharding:** When working with huge datasets, sharding, the practice of dividing bigger datasets into numerous dispersed collections, allows the database to spread and better execute queries that would otherwise be troublesome and time-consuming.
- **Load balancing:** MongoDB provides large-scale load balancing via horizontal scaling capabilities like replication and sharding. With best-in-class concurrency control and locking mechanisms that maintain data consistency, the platform can handle several concurrent read and write requests for the same data.

In summary, MongoDB provides developers flexibility and scalability thanks to the efficient horizontal scaling and load balancing capabilities. The core principles that MongoDB architecture is built on are:

- **Document data model and MongoDB Query Language:** Provides developers with a fastest way to innovate in building transactional, operational, and analytical applications.
- **Multi-cloud, global database:** Gives developers the freedom to run their applications anywhere with the flexibility to move across private and public clouds as requirements evolve – without having to change a single line of code.
- **MongoDB Cloud:** Grants a unified developer experience for modern applications that span cloud to edge, in the database, search, and the data lake, backed by comprehensive integrated application services.

MongoDB also has a project with the Bosch Group. The Bosch IoT Suite has an integrated MongoDB database and aims to change the IoT paradigm by creating services that gather data and deliver immediate insight [66].

In summary, a literature review was performed. A market research regarding existing tools whose goal was similar to the intended in this dissertation was carried out, as well as a relevant study about web applications architecture and the technologies.

Chapter 3

Work Methodology

In this chapter, the core values and principles of Agile are described. The Scrum framework and the Kanban method are also presented. The work methodology followed throughout the project is detailed.

3.1 Agile Values and Principles

In the 1990s, industry dissatisfaction led to the creation of the Agile Manifesto and the Twelve Principles of Agile Software. Many projects were canceled due to the significant time lag between business requirements and the delivery of technology meeting those demands [67].

In 2001, a group of seventeen people, later named "The Agile Alliance", gathered at The Lodge at Snowbird ski resort in Utah. It was there that the Agile Manifesto and the Twelve Principles were written in their final form [68].

The Agile Manifesto is made up of four core values and 12 supporting principles that guide the Agile software development process. The four ideas that the Agile Alliance found value in are:

- **Individuals and Interactions over Processes and Tools**
- **Working Software over Comprehensive Documentation**
- **Customer Collaboration over Contract Negotiation**
- **Responding to Change over Following a Plan**

The twelve principles of agile development include:

- Customer satisfaction by early and continuous delivery of valuable software
- Welcome changing requirements, even in late development.
- Deliver working software frequently.

- Business people and developers must work together daily throughout the project.
- Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
- Working software is the primary measure of progress.
- Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
- Continuous attention to technical excellence and good design enhances agility.
- Simplicity - the art of maximizing the amount of work not done - is essential.
- The best architectures, requirements, and designs emerge from self-organizing teams.
- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

3.2 Scrum Framework

Scrum is a lightweight agile framework that helps people, teams, and organizations generate value through adaptive solutions for complex problems [69].

Scrum is not a process, technique, or definitive method. Rather, it is a framework where it is possible to employ various processes and techniques. Scrum makes clear the relative efficacy of product management and work techniques so that can continuously improve the product, the team, and the working environment.

Scrum is built on the principles of empiricism and lean thinking. Empiricism claims that knowledge is gained via experience and decision-making based on what is seen. Lean thinking focuses on the fundamentals while reducing waste. The empirical pillars that Scrum is built on are [70]:

- **Transparency:** The emergent process and work must be visible to those performing the work as well as those receiving the work. Transparency enables inspection.
- **Inspection:** The Scrum artifacts and the progress toward agreed goals must be inspected frequently and diligently to detect potentially undesirable variances or problems. An inspection enables adaptation.
- **Adaptation:** If any components of a process wander outside of permissible boundaries, or if the finished product is undesirable, the process or materials used must be modified. To avoid future deviation, the correction must be made as soon as possible.

The successful use of Scrum depends on people improving their ability to thrive five values:

- **Commitment:** The Scrum Team commits to achieving its goals and to supporting each other.
- **Focus:** The Scrum Team's primary focus is on the work of the Sprint to make the best possible progress toward the defined goals.
- **Openness:** The Scrum Team and its stakeholders are open about the work and the challenges.
- **Respect:** Scrum Team members respect each other to be capable, independent people, and are respected as such by the people with whom they work.
- **Courage:** The Scrum Team members dare to do the right thing, to work on tough problems.

The Scrum Team is guided by these ideals in terms of their work, activities, and behavior. The decisions made, the steps done, and the manner Scrum is implemented should support rather than degrade or undermine these ideals. As they work with Scrum events and artifacts, the Scrum Team members learn and explore the values. The empirical Scrum pillars of transparency, inspection, and adaptability come to life when these values are embraced by the Scrum Team and the people they work with, generating trust.

The Scrum Team, as visible in Figure 3.1, is a small team of people that consists of one Scrum Master, one Product Owner, and Developers. There are no sub-teams or hierarchies in a Scrum Team. It is a group of professionals who work together to achieve one goal at a time, the Product Goal.

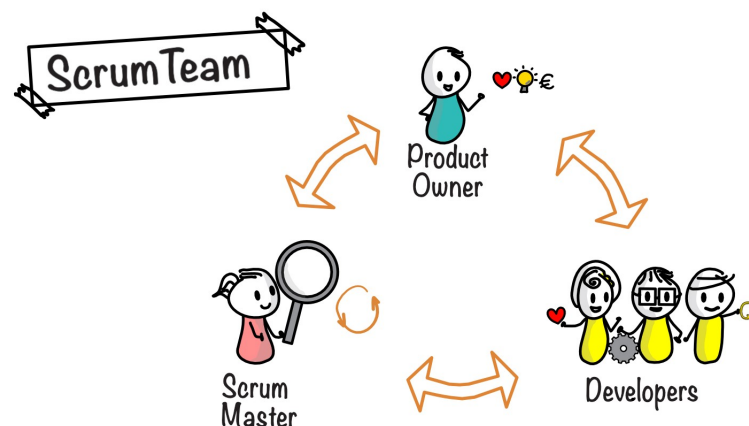


Figure 3.1: Scrum Team
[70]

- **Scrum Master:** The Scrum Master is responsible for promoting and supporting the Scrum Values. The Scrum Master helps those outside the Scrum Team understand which of their interactions with the Scrum Team are and are not helpful. The Scrum Master helps everyone change these interactions to maximize the value created by the Scrum Team.
- **Product Owner:** The Product Owner is responsible for maximizing the value of the product resulting from the work of the Developers. How this is done may vary widely across organizations, Scrum Teams, and individuals. The Product Owner is the sole person responsible for managing the Product Backlog.
- **Developers:** The Development Team, or Developers, consists of professionals who do the work of delivering a potentially releasable Increment of “Done” product at the end of each Sprint. Development Teams are structured and empowered by the organization to organize and manage their work.

Scrum Teams are cross-functional and self-organizing. Rather than being guided by individuals outside the team, self-organizing teams pick the best way to complete their tasks. Cross-functional teams have all of the skills required to complete the task without relying on individuals who are not members of the team.

The Sprint serves as a hub for all other events. Scrum provides a structured opportunity to inspect and adapt Scrum artifacts at each event. These gatherings are expressly organized to provide the necessary transparency. Any event that is not operated according to the instructions will result in missed opportunities to inspect and adapt. Events are used in Scrum to create regularity and to minimize the need for meetings not defined in Scrum.

Sprints are the heartbeat of Scrum, where ideas are turned into value. They are fixed length events of one month or less to create consistency. A new Sprint starts immediately after the conclusion of the previous Sprint. During the Sprint, no changes are made that might jeopardize the Sprint Goal, quality is maintained, the Product Backlog is fine-tuned as necessary, and as more information becomes available, the scope may be clarified and renegotiated with the Product Owner. All the work necessary to achieve the Product Goal, including Sprint Planning, Daily Scrums, Sprint Review, and Sprint Retrospective, happen within Sprints.

Sprint Planning initiates the Sprint and it's where the team determines the product backlog items that they will work on during the sprint. Sprint Planning is timeboxed to a maximum of eight hours for a one-month Sprint. For shorter Sprints, the event is usually shorter.

The Daily Scrum is a 15-minute time-boxed event for the Development Team that is held every day of the Sprint. The Developers use the Daily Scrum to inspect progress toward the Sprint Goal and to inspect how progress is trending toward completing the work in the Sprint Backlog.

The Sprint Review's goal is to examine Sprint's results and make recommendations for future changes. The Scrum Team presents their work to key stakeholders and discusses their progress toward the Product Goal. The Sprint Review is Sprint's second-to-last event, with a maximum time limit of four hours for a one-month Sprint. The event is usually shorter for shorter Sprints.

The Sprint Retrospective is used to brainstorm methods to improve quality and efficiency. The Scrum Team evaluates how the previous Sprint went in terms of people, processes, tools, and their Definition of Done. To improve its effectiveness, the Scrum Team determines the most beneficial adjustments. The most significant enhancements are implemented as quickly as possible. They could even be included in the Sprint Backlog for the following Sprint. The Sprint Retrospective brings the Sprint to a close. For a one-month Sprint, it is timed to a maximum of three hours. The event is usually shorter for shorter Sprints.

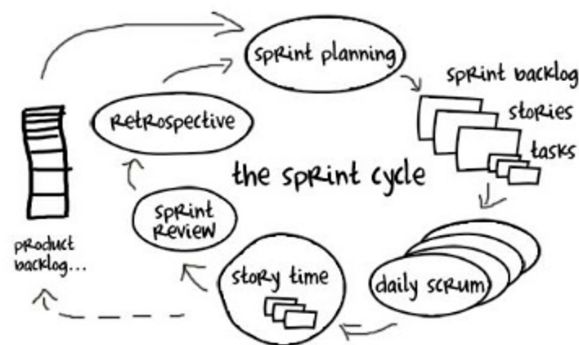


Figure 3.2: Scrum Events within a Sprint
[70]

Artifacts in Scrum represent effort or value. They're made to make crucial information as transparent as possible. As a result, everyone who inspects them has the same starting point for adaptation. Each artifact includes a promise to give information that improves transparency and focus, as well as a benchmark against which progress can be measured:

- For the Product Backlog, it is the Product Goal.
- For the Sprint Backlog, it is the Sprint Goal.
- For the Increment, it is the Definition of Done.

For the Scrum Team and their stakeholders, these commitments exist to strengthen empiricism and the Scrum values.

The Product Backlog is a constantly evolving, prioritized list of what needs to be done to improve the product. It is the Scrum Team's sole point of contact for all work. The sizing is the

responsibility of the Development Team who will be executing the work. The Product Owner can influence the Developers by assisting them in understanding and choosing trade-offs. The Product Goal is the Product Backlog commitment. It describes the product's future state and can be used by the Scrum Team to plan. The Scrum Team's long-term goal is called the Product Goal. They must complete one goal before moving on to the next.

The Sprint Backlog is made up of the Sprint Goal (why), a list of Product Backlog items chosen for the Sprint (what), and a plan for delivering the Increment (how). The Sprint Backlog is a strategy developed by and for developers. It's a highly visible, real-time representation of the work the Development Team aims to complete during the Sprint to meet the Sprint Goal. As a result, the Sprint Backlog is updated as new information becomes available throughout the Sprint. It should be detailed enough for them to check their progress in the Daily Scrum. The Sprint Goal is the Sprint Backlog commitment and it's the single objective for the Sprint.

An increment is a tangible step toward achieving the Product Goal. Each Increment is added to the previous ones and thoroughly tested, ensuring that they all work together. The Increment must be useful to offer value. Work that does not meet the Definition of Done cannot be considered part of an Increment. The Definition of Done is the Increment commitment and is a formal definition of the condition of the Increment when it meets the product's quality metrics.

The Scrum framework, as stated above, is unchangeable. While it is possible to adopt only elements of Scrum, the end outcome will not be Scrum. Scrum is a container for other approaches, methodologies, and practices, and it exists only in its entirety.

3.3 Kanban Method

Kanban is a method designed to help visualize the work, maximize efficiency and be agile. The Kanban board normally follows the structure "To Do", "In Progress" and "Done" [71].

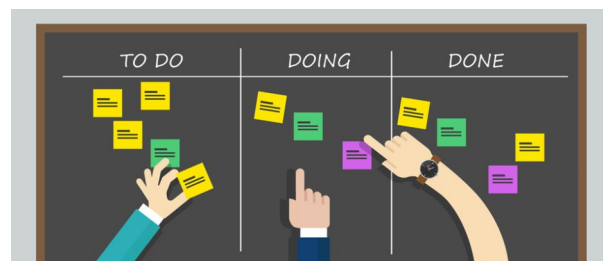


Figure 3.3: Kanban Board

The Kanban method follows two types of principles and six practices. The first principle type is the Change Management principle which is composed by:

- **Start with what you already do:** The Kanban method does not prompt you to change the process. It is based on the idea of evolving the current process.
- **Continuous improvement:** It presupposes the agreement among the members of the team to pursue constant and continuous improvement. Everyone should see their work environment as conducive to evolving and improving.
- **Respect the current process, roles, responsibilities, and positions:** It is likely that current processes have some elements that work well and are worth preserving. Fear of change must be expelled to facilitate future improvement changes.

The second principle type is the Service Delivery principle which includes:

- **Focus on customer's needs and expectations:** Delivering value to the customer should be at the center of each organization.
- **Manage the work:** Managing the work in your network of services ensures that you empower people's abilities to self-organize around the work.
- **Regularly review the network of services:** Once developed, a service-oriented approach requires continuous evaluation to foster a customer service culture.

The six practices that must be taken into account when trying to implement the Kanban method are:

- **Limit Work in Progress (WIP):** It means implementing a pull system in part or the whole process. The fundamental idea is that the work to be performed at each stage of the workflow is limited. The new work is "pulled" into a new stage of information creation.
- **Visualize Workflow:** The flow of work items through each stage of the process should be monitored and reported.
- **Make explicit process policies:** Without an explicit understanding of how things work and how the work is done, any discussion of problems tends to be emotional and subjective. With an explicit understanding, it is possible to move a more rational and objective discussion of the question - this is more likely to facilitate consensus around suggestions for improvement.
- **Improve collaboratively:** The way to achieve continuous improvement and sustainable change within an organization is through collaboratively implementing changes based on scientifically proven methods, feedback, and metrics.
- **Feedback Loops:** For teams and companies that want to be more agile, implementing feedback loops is a mandatory step. They ensure that organizations are adequately responding to potential changes and enable knowledge transfer between stakeholders.
- **Manage Flow:** The flow must be healthy and smooth. Having a smooth flow means that the risk is being minimized and the costs of delays are avoided.

3.4 Project Methodology

For this project, Scrum was used as the base framework for work package management. However, since it is being developed by one developer (Dev) only, and in accordance with the Product Owner (PO), the Bosch supervisor, some adaptations were made to better fit the needs. The sprints have a duration of 2 weeks and the following ceremonies were contemplated: Daily Meetings, Sprint Planning, Sprint Review, and Retrospective:

- Daily Meetings were not held every day, but 3 times a week. This was agreed upon by both PO and Dev after some iterations.
- Sprint Planning is the ceremony held at the beginning of the sprint, where the User Stories (US) are committed. No changes were applied to this ceremony.
- Sprint Review and Retrospective ceremonies were merged into one ceremony with two distinct parts. The first part of the ceremony consists of the Review, where both Dev and PO looked at the US committed to the sprint and checked if they were complete. The second part is the retrospective, where good and bad points for the sprint were collected and action were derived from the conclusions.
- User Stories were not voted and metrics were not used in this project due to the size of the team and the short timeline.

The Product Backlog features were defined by the PO at the beginning of the project while the User Stories were created throughout the project. The Product Goal was to have a proof of concept web application capable of meeting all the requirements desired by the client.

The main reason the scrum framework was used for the development of this project, besides the fact that it is the methodology used by Bosch for software development, was essentially its agile capacity. The fact that the software developed is delivered every 2 weeks allows the PO to give feedback on the work produced by the Dev, and together, bring it closer to the client's expectations, instead of, for example, only delivering the work at the end and not be in line with customer expectations.

3.4.1 RTC

Rational Team Concert (RTC) from IBM was the tool used to check the project's progress. User Stories were created in this tool using the interface shown in Figure 3.4.

For each US was defined a Summary, Description, and an Acceptance Criteria. Each US is then committed to a given sprint and a team. In this project, the sprints had a duration of 2 weeks.

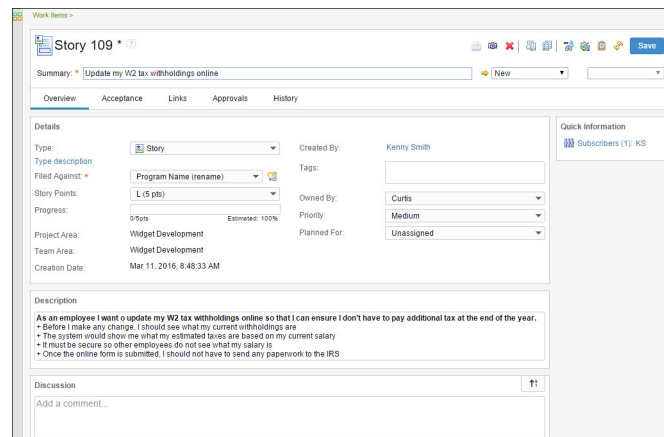


Figure 3.4: RTC User Story Example
[72]

The US had a correspondent feature as a parent, and all features had the same epic as a parent. The Epic is considered the whole Infrastructure Management Tool. The Features were created keeping in mind the value that the tool could provide to the end-user (e.g. Login, Components catalog, Search Components). A feature is then broken down into the several US, allowing to commit to smaller parts of the feature during the sprint.

RTC provides several views that allow tracking the progress of the implementation. Two views were used: roadmap and backlog. In the figure below a sprint backlog Kanban board is shown:

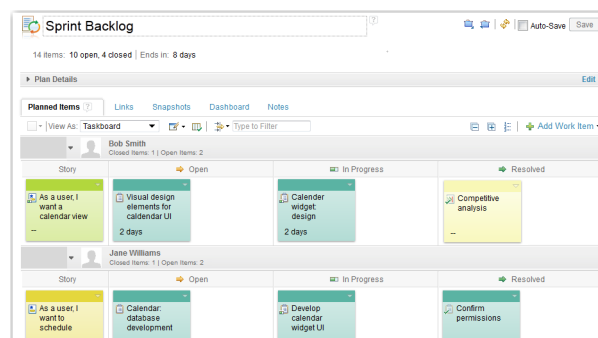


Figure 3.5: RTC Sprint Kanban Board
[73]

The Roadmap view allows to allocate the Features to a given sprint and check the backlog. Kanban view allows checking the US progress during the sprint.

In this project, the RTC was very important for the management of the work carried out. The ability to create user stories, then organize them by sprints and have all this information available in a graphic tool, made the development of the project much easier and more organized.

In summary, learning concepts such as Agile, Scrum and Kanban were essential to organize the work related to the project.

Chapter 4

System Overview and Design

In this chapter, the system requirements defined by the client are discussed. After, these requirements are successively deconstructed into a class diagram, use case diagrams, sequence diagrams, UI page mockups, and API endpoints to define the system architecture and design.

4.1 System Requirements

As mentioned in chapter 1, the system requirements were already defined by Bosch section XC-C12/ESD5. Nevertheless, these requirements were not immutable and could be changed throughout the project, if there was an agreement between the developer and the Bosch supervisor.

The initial requirements were divided into 5 different types: User Registration Roles, Asset Information Notification System, Search Catalog, Material characteristics, and User characteristics.

The User Registration Roles feature considers the possibility for a user to register, login, edit his/her information as well as the occurrence of various roles:

- The tool shall allow the registration of a new user.
- The tool shall display an interface for the user to register.
- The tool shall allow a registered user to log in.
- The tool shall display a login interface with username/email and password.
- The tool shall allow the edition of user information by the user or by admin.
- The tool shall allow setting roles to users.
- The tool shall have 3 user levels:
 - basic,

- **scrum master**,
 - **admin**.
- **Basic** level user may:
 - Edit information of components that are available (to set himself/herself as the current holder) or in his/her possession,
 - Free a held component, setting its status to available.
- **Scrum master** level may:
 - Edit his team members' information and components.
- **Admin** level user may:
 - Edit any user and components information, even components that are currently in a certain user's possession.
- There shall exist at least one **admin**.
- The tool shall set a newly registered user role to basic.

The Asset Information Notification System functionalities allow the management of components as well as the implementation of a notification system that notifies users of the status of their components:

- The tool shall allow registering and editing components and status (components include hardware and licenses, where software (SW) licenses may be, floating or dongle).
- The tool shall only allow changing fields related to the component type.
 - When adding a new component, by selecting its type, only fields specific to these components shall be available.
- The tool shall include a notification system (via email) so that a user won't forget to update his/her component's status.
 - The notification is triggered when the estimated use time is reaching the end, so the user can update this field, or make the component available.
- If the user fails to update, upon reaching the expiration time, the tool shall send a new notification to the upper-level role (**Scrum master**).
- In case the situation persists, after a set time the tool shall send a new notification to the highest level (**admin**), thus persuading users to collaborate.

The Search Catalog implements a search engine, a catalog, and a booking system:

- The tool shall include a search engine.
- The tool shall allow the user to search for material and check its status and whereabouts.
- The tool shall include the quantity of component/ type of material in search results.
- The tool shall include a material catalog.
- The tool shall have a reservation system.
- The tool shall allow the user to request a reservation for a certain material.

The Material characteristics define the different material classes of the system:

- The characteristics of an **Hardware material** shall be:
 - Team/Section
 - Current Owner
 - Current Location
 - Availability
 - Delivery Date
 - Renewal frequency (when applicable)
 - Type of component (HW, Targets, SW License - dongle or floating)
 - Cost center
 - Serial number (when applicable)
 - Description
 - Bosch Inventory number (if applicable)
 - History (Nice to have)
- The characteristics of a **target** shall be:
 - Team/Section
 - Current Owner
 - Current Location
 - Availability
 - Delivery Date
 - Type of component (HW, Targets, SW License - dongle or floating)
 - Cost center
 - Serial number (when applicable)

- Description
- Bosch Inventory number (if applicable)
- History (Nice to have)
- The characteristics of **SW license** shall be:
 - Team/Section
 - Current Owner
 - Current Location
 - Availability
 - Delivery Date
 - Utilization due date (SW licenses or when applicable)
 - Renewal frequency
 - Type of component (HW, Targets, SW License - dongle or floating)
 - Cost center
 - Serial number
 - Description
 - Expiration Date (SW licenses, if applicable)
 - Maintenance period (SW licenses, if applicable)
 - Bosch Inventory number (if applicable)
 - History (Nice to have)

The User characteristics define the user class of the system:

- The **user characteristics** shall be:
 - Name
 - Username
 - Bosch Email
 - Password
 - Team/Section
 - Role
 - Attributed components (view in the tool)
 - Upcoming components due dates (Nice to have) (view in the tool)

The first notable change that was made, is that the management of SW licenses is no longer in the scope of the project. Instead, a new class named Cables was defined. The HW material is now called Components and targets are now a specific type of Component. The logic behind the management of Components and targets is the same while the Cables class has a different reasoning. Considering this, the way the users interact with this class is also different.

The notification system is no longer only by email but is also builtin in the tool. Three different types of notifications were also defined:

- Check Status Notification: When a user is using a component for more than 2 months he/she is notified.
- Estimated Time Notification: When the due date of a component is within a week, the user is notified.
- Over Time Notification: When the due date of a component is expired, the user, scrum master, and admin are notified.

The History feature was also added. The objective of this functionality is to track the changes made to any component or target as well as the components or targets that belong to a user.

Other minor changes were performed throughout the project leading to the final requirements.

The User Registration Roles feature main differences is how the roles interact with the Cables management:

- The tool shall allow the registration of a new user.
- The tool shall display an interface for the user to register.
- The tool shall allow a registered user to log in.
- The tool shall display a login interface with username/email and password.
- The tool shall allow setting roles to users.
- The tool shall allow the edition of the user information.
- The tool shall display an interface for the user to edit his information.
- The tool shall have 3 user levels
 - **basic,**
 - **scrum master,**
 - **admin**

- **Basic** level user may:
 - Edit his user information.
- **Scrum master** level may:
 - Edit his information and his team members' information.
- **Admin level** user may:
 - Edit any user information including changing roles.
- There shall exist at least one **admin**.
- The tool shall set a newly registered user role to basic.

The Asset Information Notification System includes now the management of cables and new types of Notifications:

- The tool shall allow registering, editing, and deleting Components and Cables.
- The tool shall display an interface for adding and editing Components and Cables as well as an option to delete them.
- **Basic** level user may:
 - Edit information of components that are in his/her possession.
 - Edit information of available components that belong to his/her team, to set himself/herself as the current holder.
 - Add cables for his/her section.
 - Edit cables that belong to his/her section.
- **Scrum master** level may:
 - Edit information of components that are in his/her possession or belong to his/her team.
 - Edit information of available components that belong to his/her team, to set himself/herself as the current holder or to set any team member to current holder.
 - Add components for his/her team.
 - Delete components that belong to his/her team.
 - Add, edit and delete cables for his/her section.
- **Admin level** user may:
 - Add, edit and delete any component or cable.

- The tool shall include a notification system (via push notifications and built-in in the tool).
- There are 3 types of notifications:
 - Check Status: The notification is triggered when the user is using the component for more than 2 months.
 - Estimated Time: The notification is triggered when the estimated use time is reaching the end.
 - Over Time: The notification is triggered when the estimated use time expired and is also sent to the upper-level roles (**Scrum master** and **admin**).
- The tool shall allow the user to acknowledge or clear a notification.
- The tool shall allow the user to acknowledge all notifications except overtime notifications.

The Search Catalog implements now an improved catalog:

- The tool shall include a search engine.
- The tool shall include a Components catalog, Targets catalog, Cables catalog, and a User Components/Targets catalog.
- The tool shall allow the user to search any material in the respective catalog for id or description/equipment.
- The tool shall allow filters to be applied in the catalog.
- The catalog shall have the option to order and sort the elements.
- The tool shall have a reservation system - now defined as Nice to Have.

The History feature tracks the changes made to any component or target and user components and targets use history:

- The tool shall display an interface with all components and targets that belonged to the user.
- The tool shall display an interface with all changes made to a component or target.

The Material characteristics define the different material classes of the system:

- The characteristics of a **Component** shall be:
 - SerialID
 - Owner
 - Location
 - Status

- Type
 - Team
 - Starting Date
 - Due Date
 - Cost
 - Equipment
- The characteristics of a **Target** are the same as a Component but the type is defined as **Target**.
- The characteristics of **a set of Cables** shall be:
 - Section
 - Description
 - Total Quantity
 - Location
 - Quantity in Use
 - Quantity Available
- The **User** characteristics define the user class of the system:
 - Name
 - Username
 - Email
 - Password
 - Team
 - Section
 - Role

4.2 Class Diagram

After analyzing carefully the final requirements the system class diagram was built as shown in Figure 4.1.

The class User defines the user characteristics described above in the system requirements. It manages all operations related to a User. The User attributes are Id, Name, Username, Email, Password, Role, Section, and Team. The User methods are Login, Signup, Edit User Information, and Get User Information. The classes Section, Role, and Team are part of the User class and help define the respective attributes. A User can have zero or more Components, zero or more Notifications, and zero or more History.

The class Component defines the component characteristics described above in the system requirements. It manages all operations related to a Component. The Component attributes are Id, Location, Serial ID, Type, Equipment, Cost, LastUpdate, StartDate, DueDate, AckDate, User, Team, and Status. The Component methods are Add Component, Edit Component, Delete Component, Get Components, Get Components By Id, and Get Components by User. The classes Status, Role, and Team are part of the Component class and help define the respective attributes. A Component can have zero or one User, zero or more History and zero or more Notifications.

The class Target defines the target characteristics described above in the system requirements. It manages all operations related to a Target. A Target is a type of Component. It shares the methods Add, Edit, Delete and also has the methods Get Targets and Get Targets by Id.

The class Cables defines the cables characteristics described above in the system requirements. It manages all operations related to a set of Cables. The Cables attributes are Id, TotalQuantity, UseQuantity, AvailableQuantity, Location, Description, and Section. The Cables methods are Add Cables, Edit Cables, Delete Cables, and Get Cables. The class Section is part of the Cables class and helps define the respective attribute. A Component can have zero or one User, zero or more History, and zero or more Notifications.

The class Notification defines the notification characteristics regarding the system requirements. It manages all operations related to a Notification. The Notification attributes are Id, Type, Date, Title, Body, User, and Component. The Notification methods are Ack Notification, AckAll Notifications, Clear Notification, and Get User Notifications. A Notification has a User associated. A Notification lives and dies with one Component.

The class History defines the history characteristics regarding the system requirements. It manages all operations related to a line of History. The History attributes are Id, InUse, EditedBy, Description, EndDate, StartDate, User, and Component. The History methods are Get Components History and Get User Components History.

The classes Role and Team describe the predefined roles and teams, respectively. They are both parts of one User or one Component.

The class Section describes the predefined sections. It is part of one User and a set of Cables.

The class Status describes the predefined status. It is part of one Component.

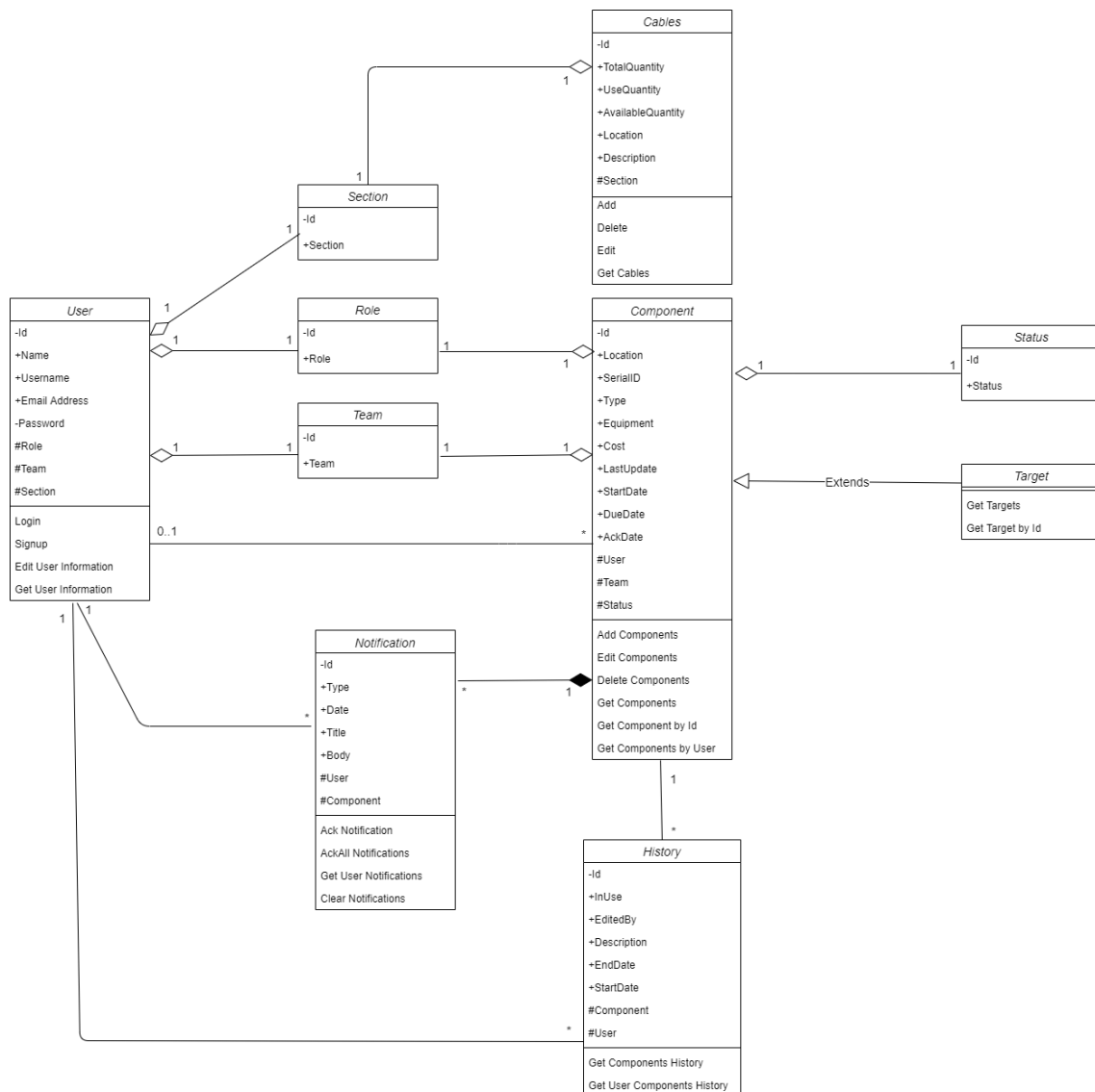


Figure 4.1: System Class Diagram

4.3 Use Case Diagrams

After reviewing the final requirements it is visible that there are three types of possible user roles: Basic, Scrum Master, and Admin. It is also possible to identify which actions a user can perform taking into consideration his role. Thus, a use case diagram was designed for each role. It's important to refer that the use cases of Admin and Scrum Master roles include the use cases of the Basic user.

4.3.1 Basic

List of use cases for the Basic user:

- Signup
- Login
- Edit own User information
- Edit information of available Components
- Edit information of available Targets
- Edit information of Components that are in his possession
- Edit information of Targets that are in his possession
- Add Cables
- Edit Cables
- Acknowledge of all his Notifications
- Clear his check status Notifications
- Acknowledge his check status Notifications
- Clear his estimated time Notifications
- Acknowledge his estimated time Notifications
- Clear his overtime Notifications
- Acknowledge his overtime Notifications

In the Figure 4.2, the use case diagram for the Basic user is shown.

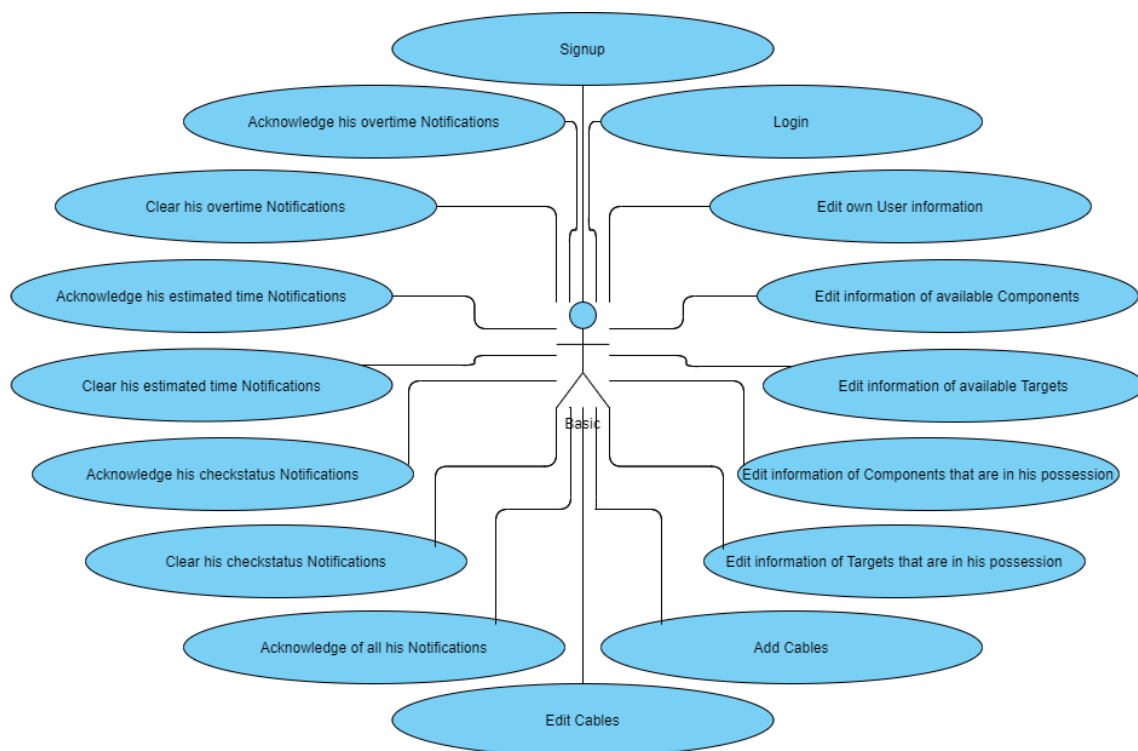


Figure 4.2: Basic User Use Case Diagram

4.3.2 Scrum Master

Besides the Basic use cases, the Scrum Mater use cases are:

- Acknowledge Team members overtime Notifications
- Clear Team members overtime Notifications
- Edit Team members User information
- Add Components for his Team members
- Delete Components of his Team members
- Delete Cables
- Edit information of Components of his Team members
- Edit information of Targets of his Team members
- Delete Targets of his Team members
- Add Targets for his Team members

In the Figure 4.3, the use case diagram for the Scrum Master user is shown.

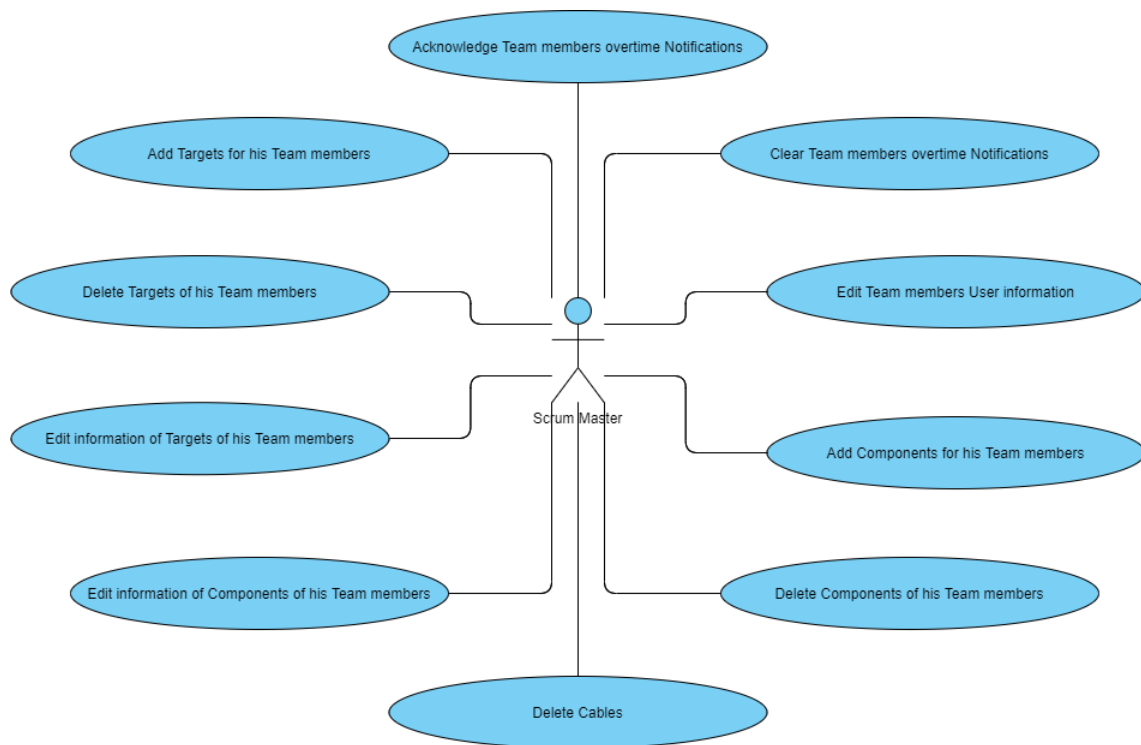


Figure 4.3: Scrum Master Use Case Diagram

4.3.3 Admin

Besides the Basic use cases, the Admin use cases are:

- Acknowledge all Users overtime Notifications
- Clear all Users overtime Notifications
- Edit all Users User information
- Add Components
- Delete any Component
- Delete Users
- Edit information of all Components
- Edit information of all Targets
- Delete any Target
- Add Targets

In the Figure 4.4, the use case diagram for the Admin user is shown.

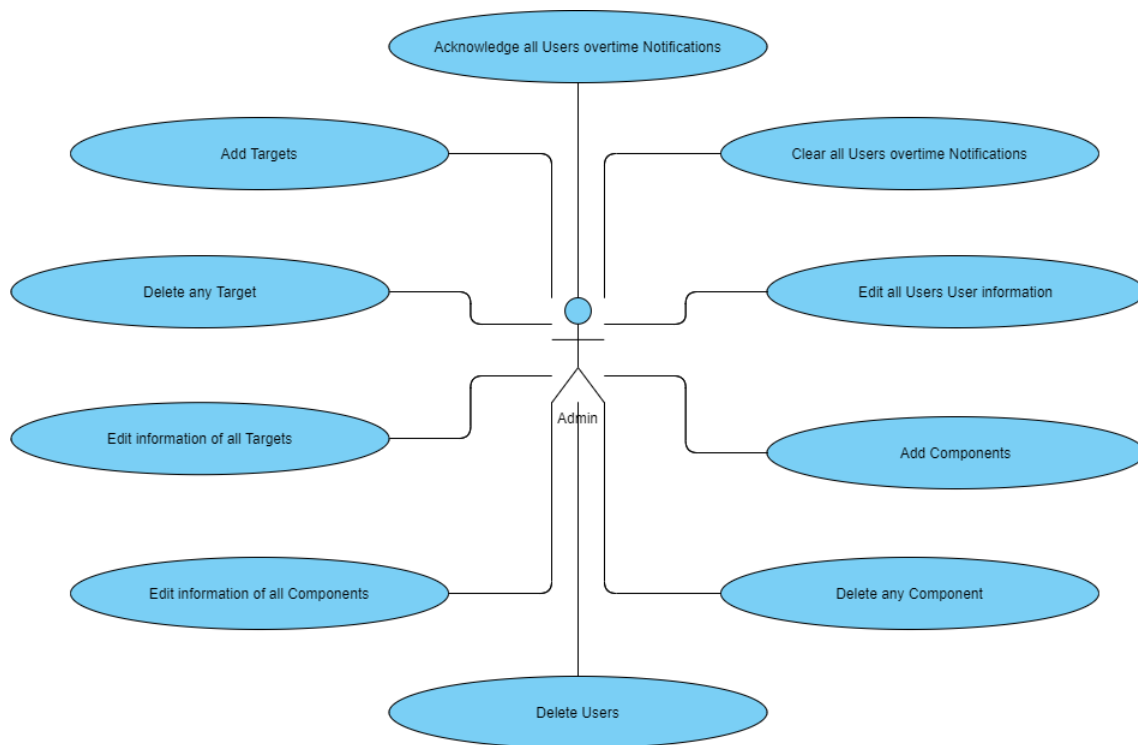


Figure 4.4: Admin Use Case Diagram

In summary, Scrum Master and Admin are upper-level roles of Basic, and Admin is an upper-role of Scrum Master. The Scrum Master can manage all information related to the team he belongs while the Admin can manage all existing data.

4.4 Sequence Diagrams

Now that the use case diagrams are set, it is necessary to design the respective sequence diagram for each use case. Even though the use case diagrams were defined for each existing role, the sequence diagrams will be composed for any user, not taking into consideration his role.

This is because these sequence diagrams are an abstraction of level 1. Therefore, there is no need to go into too much detail, such as authentication checks. Another reason is that, although a Scrum Master can, for example, edit other users' information and a Basic user can only edit his/her, the use case is very similar and the logical principle is the same, there is no need to differentiate these details.

In summary, all the existing use cases will be described with no distinction between roles.

4.4.1 Signup

The user accesses the system Signup page. The system returns the Signup page where a registering form is displayed with the following required fields: Name, Username, Email, and Password. The user fills the form and tries to register his information. The system checks if the information is valid and, in that case, saves the user details and redirects to the Login page. If the validation fails, the system informs the user with an error message. The described sequence diagram is presented in Figure 4.5.

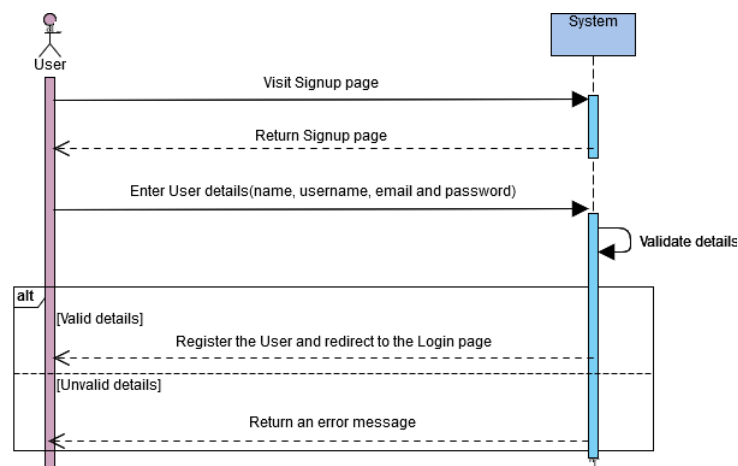


Figure 4.5: Signup Sequence Diagram

4.4.2 Login

The user accesses the system Login page. The system returns the Login page where a login form is displayed with the following required fields: Email and Password. The user fills the form and tries to login into the system. The system checks if the information is valid and, in that case,

logins the user and redirects to the Main page. If the validation fails, the system informs the user with an error message. The Login sequence diagram is presented in Figure 4.6.

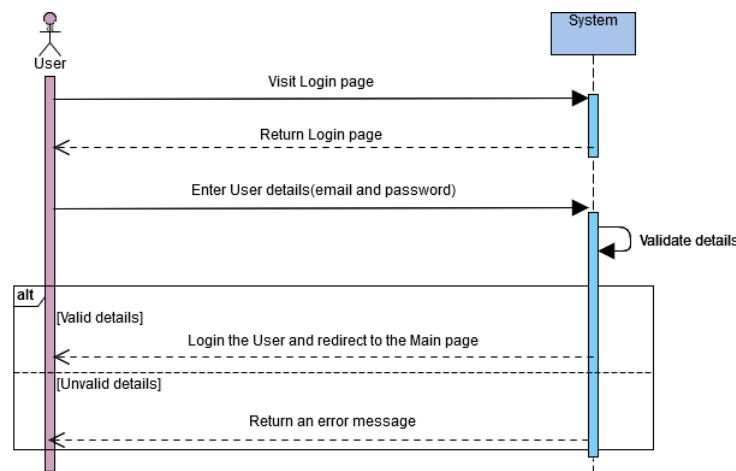


Figure 4.6: Login Sequence Diagram

4.4.3 Edit User Information

After logging into the system the User, on the Main page, accesses the User information feature. Since it is a SPA system, the system updates the Main page, displaying the User Information specific UI components. In this case, it is an editable form with the respective user information where the user can edit the username and/or password. After submitting the changes, the system validates the information (e.g. password has minimum length) and returns an error message whether the operation is invalid or saves the edited user information otherwise. The described sequence diagram is presented in Figure 4.7.

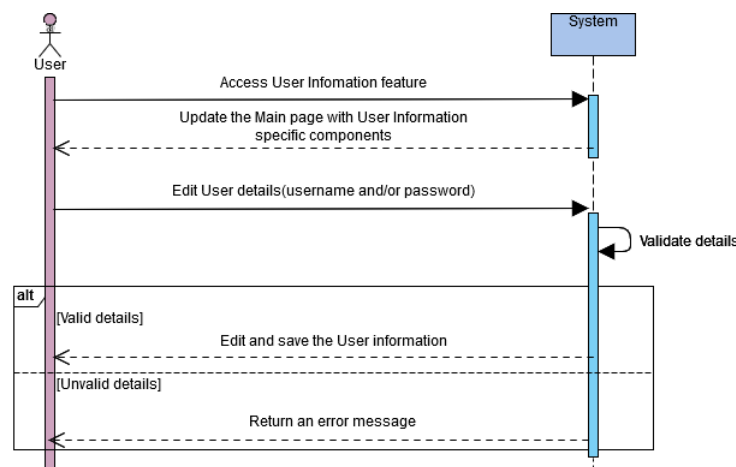


Figure 4.7: Edit User Information Sequence Diagram

4.4.4 Add Component

After logging into the system the User, on the Main page, accesses the Components Catalog. The system updates the Main page displaying the Components Catalog components. The user clicks on add component button and the system displays an empty form with the following fields: Team, User, Type, Status, Cost, and Equipment. After filling in the form, the system validates the information and saves the Component details if the validation is successful. If not, the system returns an error message. The described sequence diagram is presented in Figure 4.8.

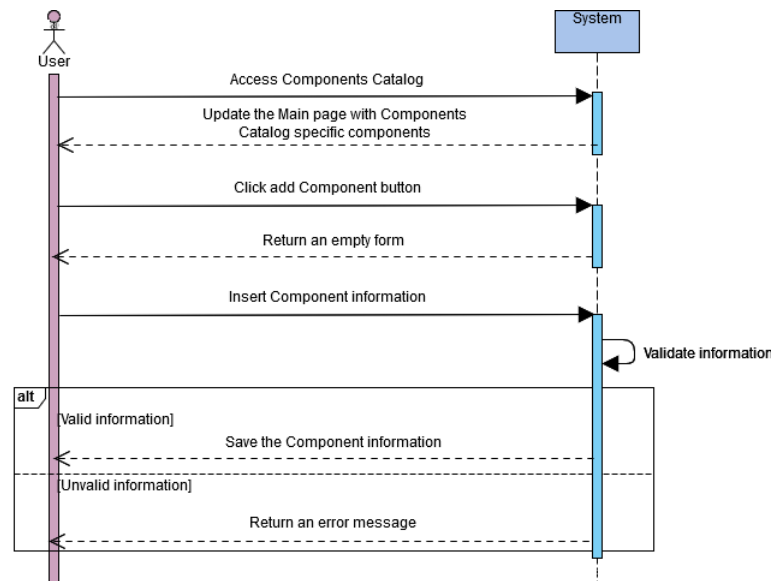


Figure 4.8: Add Component Sequence Diagram

4.4.5 Edit Component

After logging into the system the User, on the Main page, accesses the Components Catalog. The system updates the Main page displaying the Components Catalog components. The user selects a component to edit and the system displays an editable form with the component information. After submitting the changes, the system validates the information and saves the new Component details if the validation is successful. If not, the system returns an error message. The Edit Component information sequence diagram is presented in Figure 4.9.

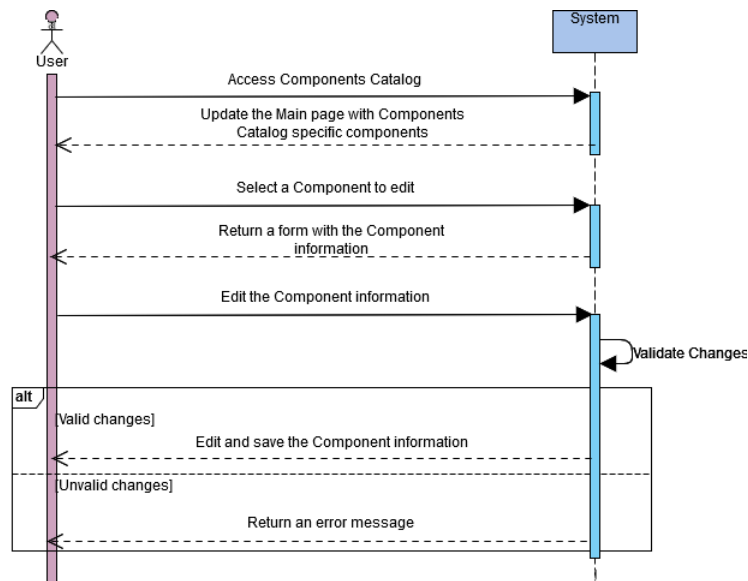


Figure 4.9: Edit Component Sequence Diagram

4.4.6 Delete Component

After logging into the system and accessing the Components Catalog the User selects one or more components that desires to delete. The system returns a confirmation modal where the User can choose to confirm his decision or cancel. If the User selects the confirm button the components are deleted from the system. If not, the confirmation modal is closed. The Delete Component sequence diagram is presented in Figure 4.10.

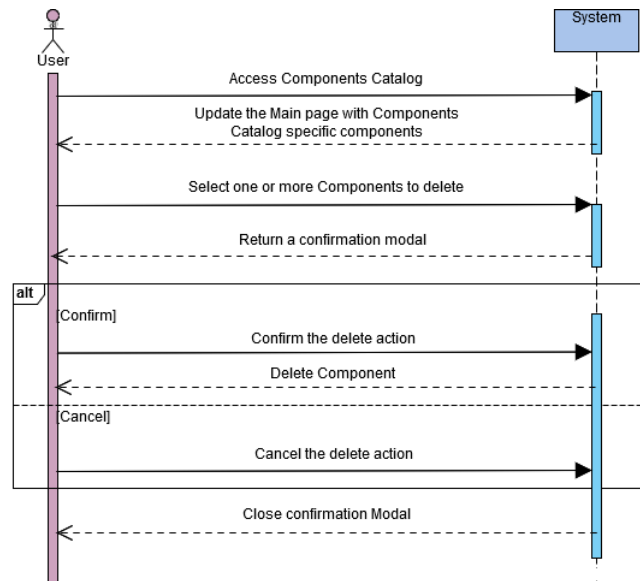


Figure 4.10: Delete Component Sequence Diagram

4.4.7 Add Target

After logging into the system the User, on the Main page, accesses the Components or Targets Catalog. The system updates the Main page displaying the chosen Catalog components. The user clicks on add component or target button and the system displays an empty form with the field Type non-editable and with the predefined value Target. After filling in the form, the system validates the information and saves the Target details if the validation is successful. If not, the system returns an error message. The described sequence diagram is presented in Figure 4.11.

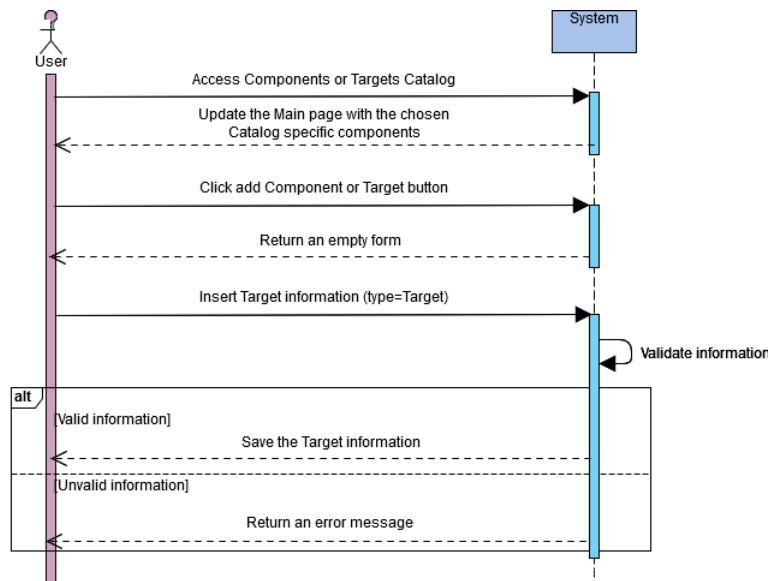


Figure 4.11: Add Target Sequence Diagram

4.4.8 Edit Target

After logging into the system the User, on the Main page, accesses the Components or Targets Catalog. The system updates the Main page displaying the chosen Catalog components. The user selects a Target to edit and the system returns an editable form. After the User submits the changes, the system validates the information and saves the Target details if the validation is successful. If not, the system returns an error message. The Edit Target sequence diagram is presented in the Figure 4.12.

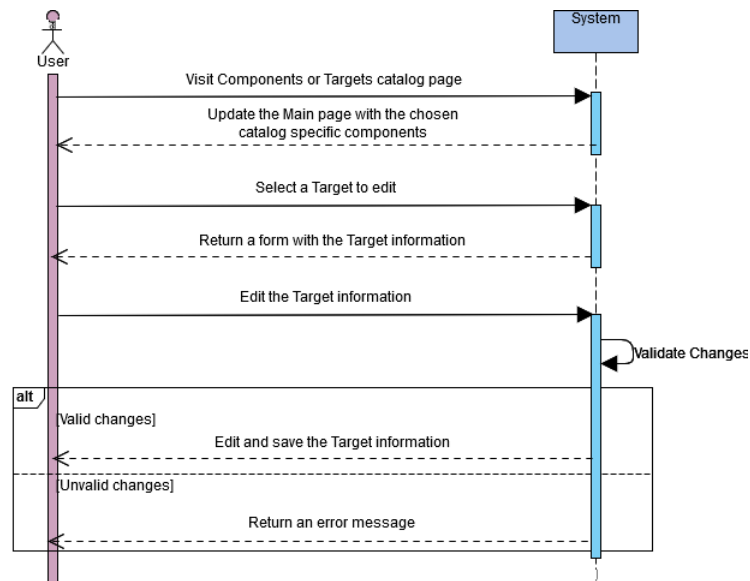


Figure 4.12: Edit Target Sequence Diagram

4.4.9 Delete Target

After logging into the system the User, on the Main page, accesses the Components or Targets Catalog. The system updates the Main page displaying the chosen Catalog components. The user selects one or more Targets to delete. The system returns a confirmation modal where the User can choose to confirm his decision or cancel. If the User selects the confirm button the targets are deleted from the system. If not, the confirmation modal is closed. The Delete Target sequence diagram is presented in Figure 4.13.

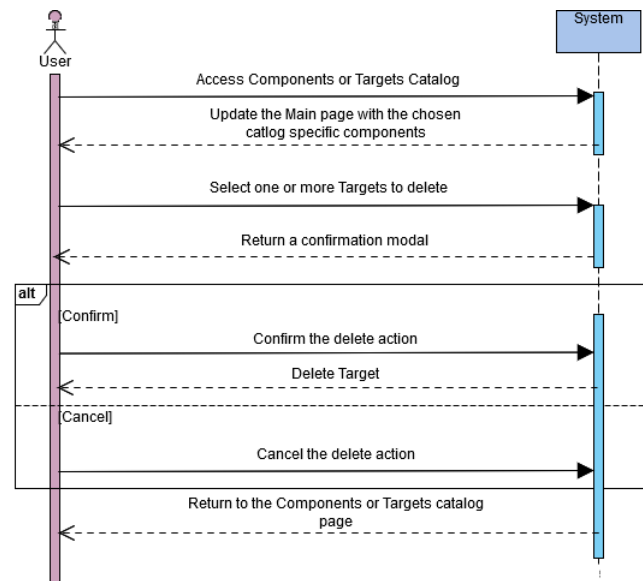


Figure 4.13: Delete Target Sequence Diagram

4.4.10 Add Cables

After logging into the system the User, on the Main page, accesses the Cables Catalog. The system updates the Main page displaying the Cables Catalog components. The user clicks on add cables button and the system displays an empty form with the following fields: Section, Description, Quantity, Location, Available Quantity, and In-Use Quantity. After filling in the form, the system validates the information and saves the Cables details if the validation is successful. If not, the system returns an error message. The Add Cables sequence diagram is presented in Figure 4.14.

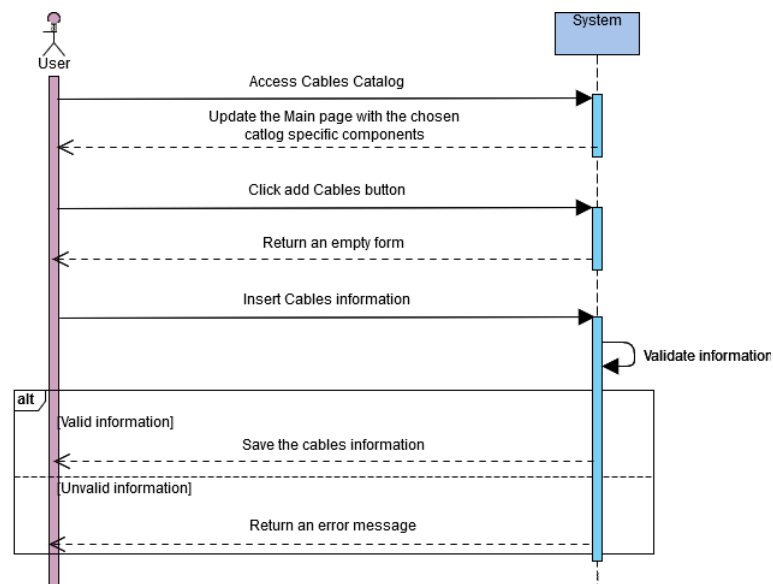


Figure 4.14: Add Cables Sequence Diagram

4.4.11 Edit Cables

After logging into the system the User, on the Main page, accesses the Cables Catalog. The system updates the Main page displaying the Cables Catalog components. The user selects a set of cables to edit and the system displays an editable form with the cables information. After submitting the changes, the system validates the information and saves the new set of Cables detail if the validation is successful. If not, the system returns an error message. The Edit Cables sequence diagram is presented in Figure 4.15.

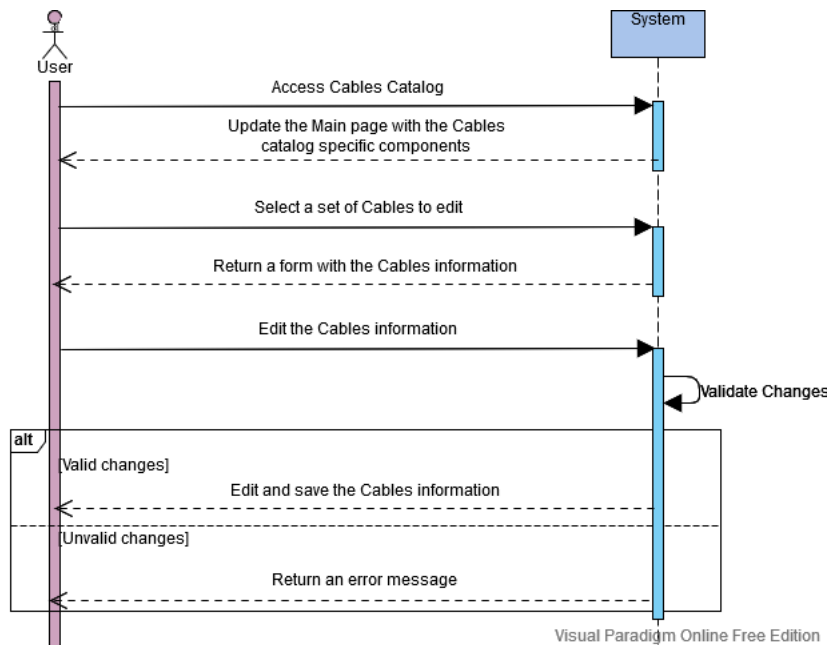


Figure 4.15: Edit Cables Sequence Diagram

4.4.12 Delete Cables

After logging into the system and accessing the Cables Catalog the User selects one or more components that desires to delete. The system returns a confirmation modal where the User can choose to confirm his decision or cancel. If the User selects the confirm button the cables are deleted from the system. If not, the confirmation modal is closed. The Delete Cables sequence diagram is presented in Figure 4.16.

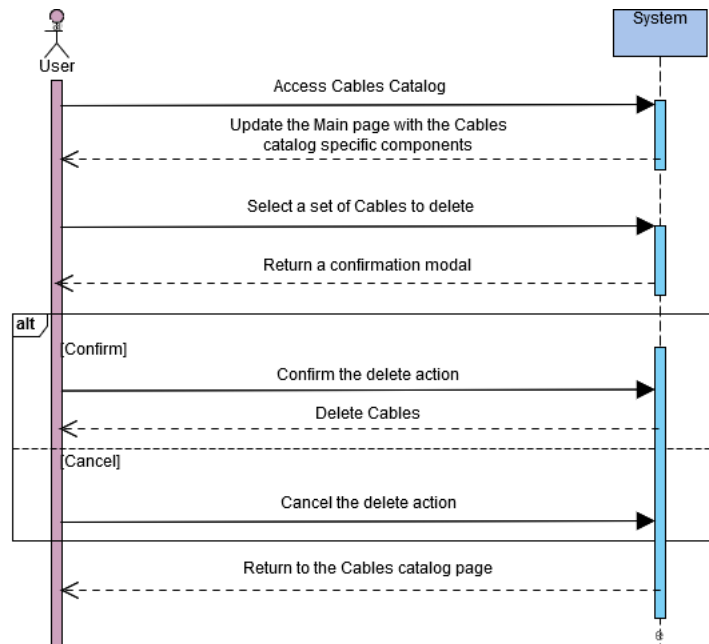


Figure 4.16: Delete Cables Sequence Diagram

4.4.13 Acknowledge Notification

After logging into the system, the User opens the notifications menu. The system returns a list of notifications regarding the respective User. The User clicks on the notification acknowledge icon and the system automatically refreshes the list and updates the component related to that notification. This option is only able in notifications of the type estimated time or check status. The Acknowledge Notification sequence diagram is presented Figure 4.17.

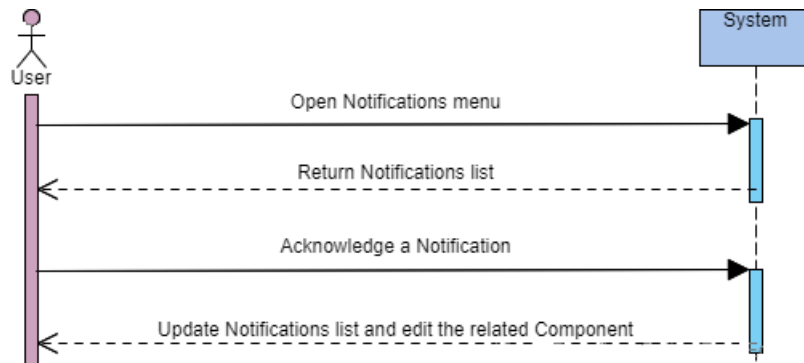


Figure 4.17: Acknowledge Notification Sequence Diagram

4.4.14 Clear Notification

After logging into the system, the User opens the notifications menu. The system returns a list of notifications regarding the respective User. The User clicks on the notification clear icon and the system returns a confirmation modal. If the User selects to continue, the system automatically refreshes the list and updates the component related to that notification. If not, nothing happens. In both cases, the system closes the confirmation modal. This option is only able in notifications of the type estimated time or check status. The Clear Notifications sequence diagram is presented in Figure 4.18.

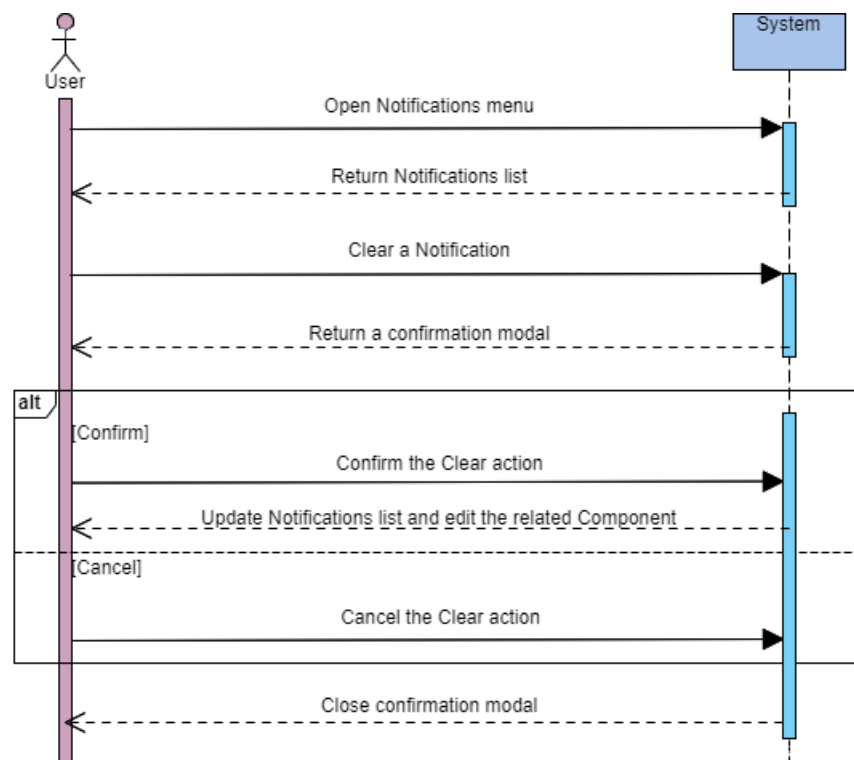


Figure 4.18: Clear Notification Sequence Diagram

4.4.15 Acknowledge All Notifications

After logging into the system, the User opens the notifications menu. The system returns a list of notifications regarding the respective User. The User selects the option to acknowledge all notifications and the system automatically refreshes the list and updates the components related to the existing notifications. This option is only applied to notifications of the type estimated time or check status. The Acknowledge All Notifications sequence diagram is presented in Figure 4.19.

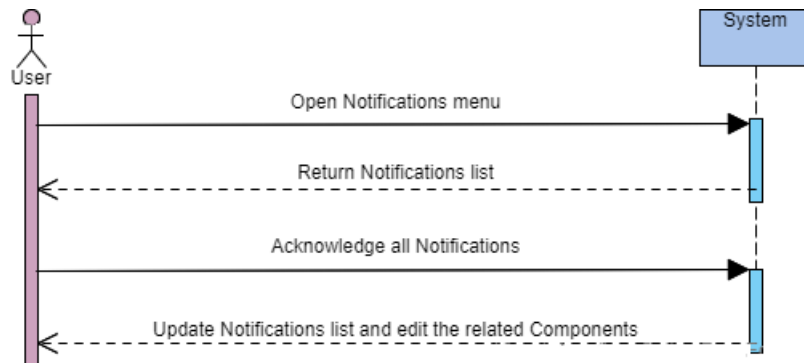


Figure 4.19: Acknowledge All Notifications Sequence Diagram

4.4.16 Overtime Notification

After logging into the system, the User opens the notifications menu. The system returns a list of notifications regarding the respective User. The User clicks in an overtime notification and the system returns a confirmation modal. The User can choose to acknowledge the notification or to clear it. In both cases, the system closes the confirmation modal, refreshes the notifications list, and updates the component related to the notification. The Overtime Notification sequence diagram is presented in Figure 4.20.

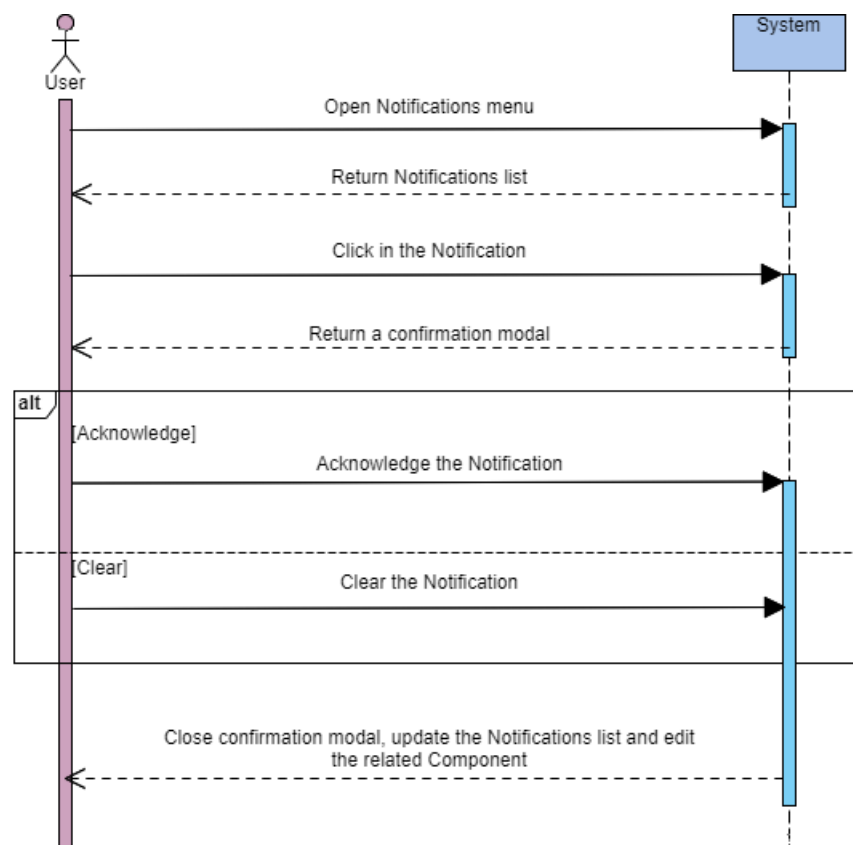


Figure 4.20: Overtime Notification Sequence Diagram

4.5 User Interface Mockup and Structure

After establishing the main use cases and understanding the user's interactions with the system, it is important to determine the graphical interface with which the user will interact. For this, a simple mockup of the application UI was produced.

Before the user interface design, the structure of the application's pages was defined, as shown in Figure 4.21.

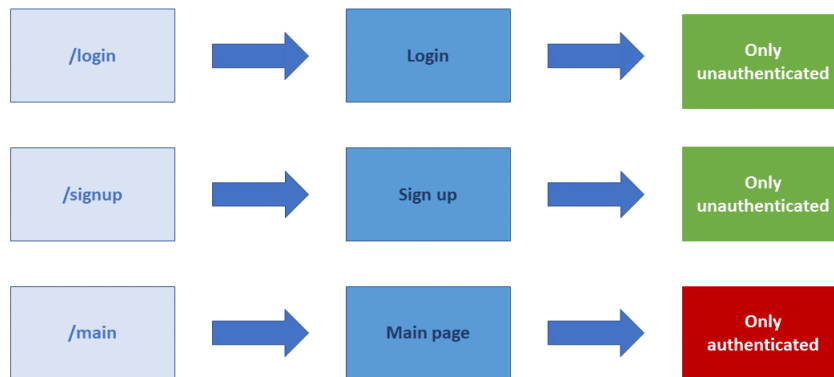


Figure 4.21: Application Pages

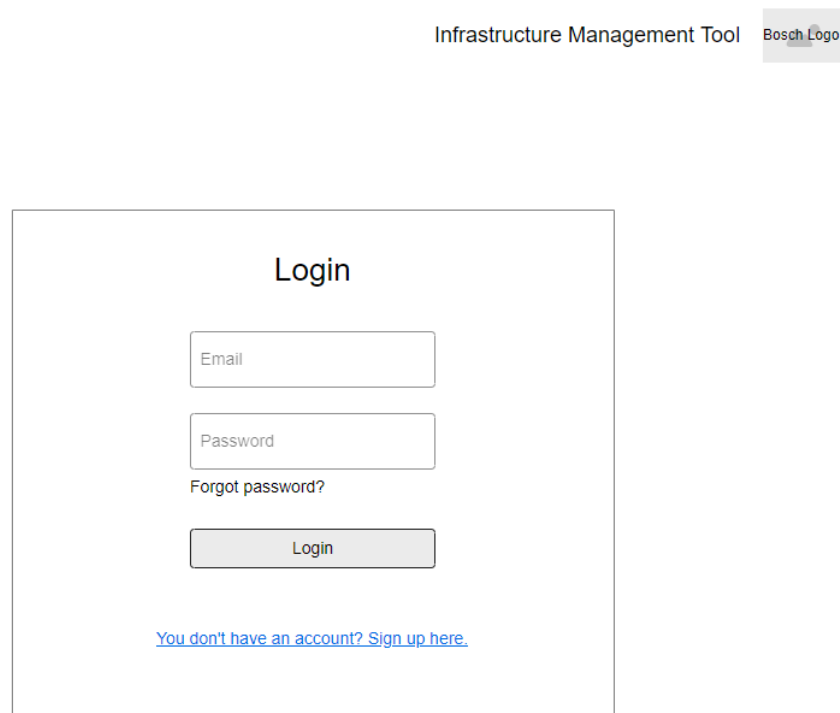
As mentioned in chapter 2, this web application is based on the SPA architecture although it appears to have several "pages". That feeling is because, thanks to the react-router-dom, which is described in the next chapter, it renders react components based on a path the user enters into the URL of the browser. In the end, we are always on the same page which renders the react components related to the path chosen by the user. The three defined paths are:

- /login, where the user must be unauthenticated and can log into the application, becoming authenticated.
- /signup, where the user must be unauthenticated.
- /main, where the user must be authenticated and can log out of the application, becoming unauthenticated.

The logic behind user authentication is described in the system implementation chapter 5.

4.5.1 /login

The login page is the application start web page. It allows the user to redirect to the /signup page if the user is not yet registered or to log in to the application by inserting his credentials. After logging in, the user is redirected to the /main page. The Figure 4.22 represents the proposed UI.



The mockup shows a web page for the 'Infrastructure Management Tool'. The header includes the application name and the Bosch logo. The main content area is titled 'Login' and contains a form with an 'Email' input field, a 'Password' input field, a 'Forgot password?' link, and a 'Login' button. Below the form is a link that says 'You don't have an account? Sign up here.'

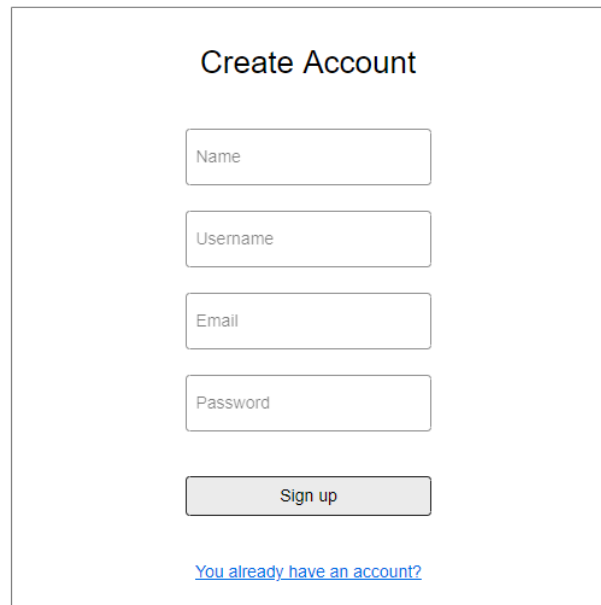
Figure 4.22: Login UI Mockup

The page is composed of a header with the application name and Bosch logo, a login form and submit button, and a hyperlink text to the /signup page.

4.5.2 /signup

The signup page is responsible for displaying a registration form to the user. The user can also be redirected to the /login page if he/she is already registered. The Figure 4.23 represents the signup page proposed UI.

The page is composed of a header with the application name and Bosch logo - same as in the /login page, a registration form and submit button, and a hyperlink text to the /login page.



The mockup shows a 'Create Account' form with the following elements:

- Title: **Create Account**
- Input fields: Name, Username, Email, Password
- Button: **Sign up**
- Link: [You already have an account?](#)

Figure 4.23: Sign up UI Mockup

4.5.3 /main

The main page is the main application page. The user needs to be authenticated to access this page. This page is essentially composed of two "static" elements: the header and the sidebar, and a dynamic element - the space that is not covered either by the header and the sidebar.

The sidebar works as a navigation bar that lets the user navigate through the Components catalog, Targets catalog, MyComponents catalog, Cables catalog, and the Logout option.

The header displays the application name, Bosch logo, user name, and user username. It also allows the user to check the notifications menu, User Information content and to minimize the sidebar.

In the remaining space, the components are re-rendered every time the user navigates to another "page". Figure 4.24 represents the proposed UI for the main page:

The Figures A.5, A.6, A.7, A.8, A.9 show the remaining application proposed design.

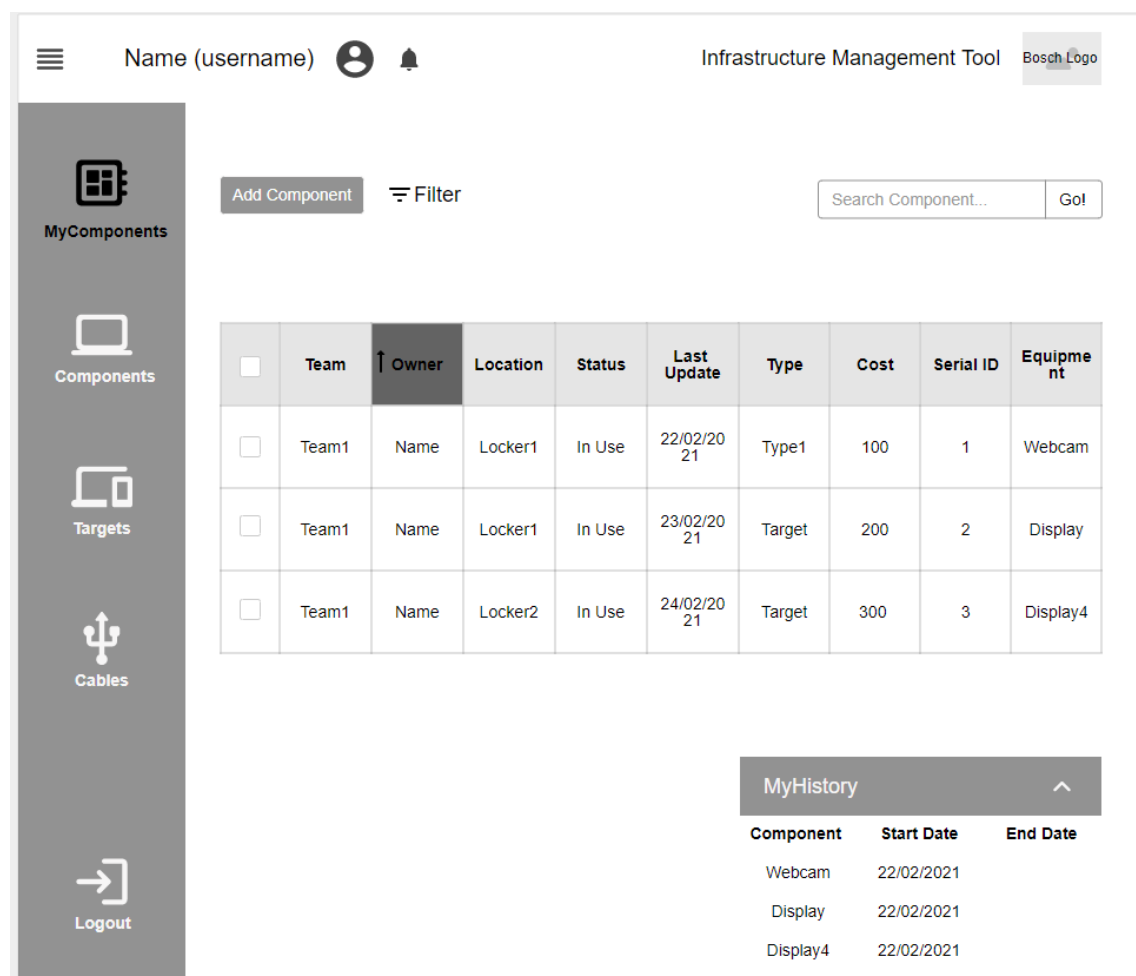


Figure 4.24: Main UI Mockup

4.6 Backend API Endpoints

Now that the frontend UI is sketched, it is possible to start defining the backend endpoints. As stated in chapter 2, the REST API was the architectural style chosen to build the interactions between the user and the backend system.

If we look into the class diagram presented in section 4.2, it's visible that the system contains ten data models. Thus, we'll have at least ten main endpoint areas, where each request represents the method defined in the respective class model.

Each API call needs an endpoint, which is the URL that is requested. The root-endpoint and path of an endpoint determine the resource which is being requested. The method is the next component required for the REST API call. In the REST API, there are four most typically used methods:

- GET - get data.

- POST - create data.
- PATCH - update data.
- DELETE - delete data.

In Figure 4.25 all API endpoints that were defined are presented.

In summary, the system is ready to be implemented. The final requirements of the project were defined and consequently the classes and use cases of the system were designed. Then, taking into account the existing information, the interaction of a user with the system was defined at a simpler level. Finally, a mockup of the UI was designed and the backend endpoints defined. Thus, the implementation phase is exclusively responsible for the software development, since the application architecture and design are already defined.

/users/...		/components/...	
GET .../ Retrieve list of all users	POST .../signup Create new user	POST .../ Retrieve list of all componentes with filters applied	POST .../user Retrieve list of all componentes for a specific user
POST .../login Log user in	POST .../user Retrieve user information	POST .../id Retrieve component information	POST .../new Create new component
PATCH .../edit Update a user	DELETE .../delete Delete a user	PATCH .../edit Update a component	DELETE .../delete Delete one or more components
/notifications/...		/cables/...	
POST .../user Retrieve list of all notifications for a specific user	POST .../ack Acknowledge a check status/estimated time notification	POST .../ Retrieve list of all cables with filters applied	POST .../id Retrieve cables information
POST .../ackot Acknowledge a a overtime notification	POST .../clear Clear a notification	PATCH .../edit Update a set of cables	POST .../new Create new cables
POST .../ackall Acknowledge all check status/estimated time notifications		DELETE .../delete Delete a set of cables	
/history/...		/targets/...	
POST .../user Retrieve list of all used components for a specific user	POST .../component Retrieve a list of all changes made to a component	POST .../ Retrieve list of all targets with filters applied	POST .../new Create new component
/sections/...	/status/...	POST .../id Retrieve target information	DELETE .../delete Delete one or more targets
GET .../ Retrieve list of all sections	GET .../ Retrieve list of all status	PATCH .../edit Update a target	
/teams/...	/roles/...		
GET .../ Retrieve list of all teams	GET .../ Retrieve list of all roles		

Figure 4.25: API Endpoints

Chapter 5

System Implementation

In this chapter, the system setup is presented, the third-party libraries integrated are described, and the features developed are detailed.

5.1 System Setup

The technologies used to develop this application belong to the MERN stack. MERN stands for MongoDB, Express, React, and Node.js, and it is an architecture that allows the development of a 3-tier architecture (frontend, backend, database) entirely using JavaScript and JSON. React is used to build the client-side of the application while Node and Express are responsible for the server-side. MongoDB is used to store the application data.

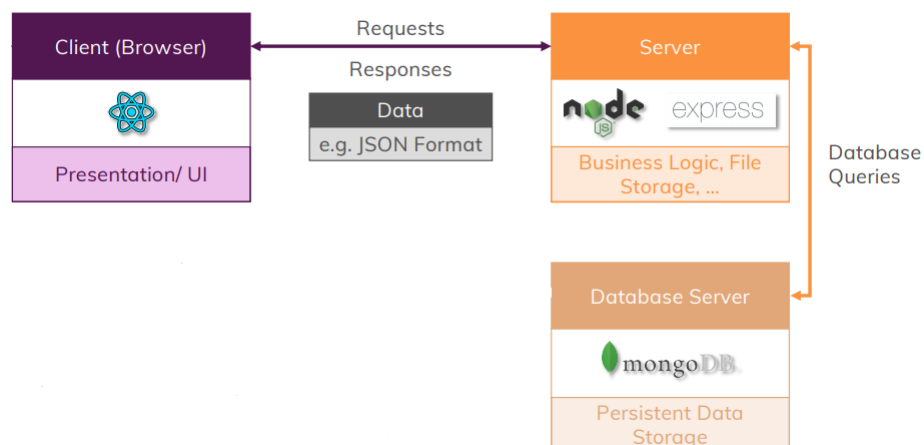


Figure 5.1: MERN Stack
[74]

As seen in Figure 5.1, the client-side communicates with the server-side using HTTP requests, and the server, in turn, sends queries to the database. The client doesn't communicate directly to the database.

To send queries to the database, it is necessary to include the database credentials, username, and password, in the code. In a browser, every user has access to all the code that runs on the client-side. It would be a security breach if any person could access the database credentials so easily. To solve this issue, it's the server-side, where the code is not readable by any user, that is responsible for sending queries to the database. In summary, the users send requests to the server from the client-side, and the server-side decides which requests to handle and which responses want to send back. The server-side has full control of the application.

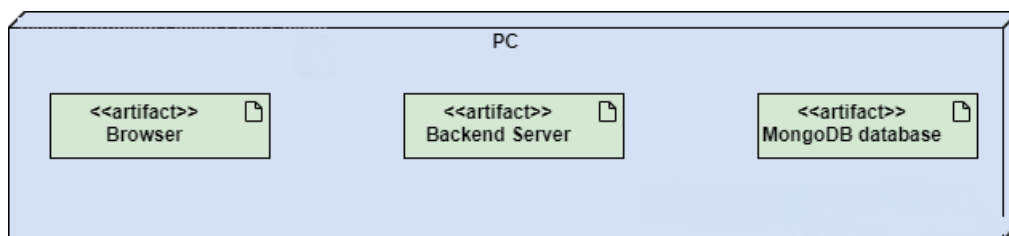


Figure 5.2: Deployment Diagram

In terms of deployment, as seen in the Figure 5.2, both the browser, the backend server and the database are running on the same machine.

5.2 Third-party Libraries Integration

Now that the system deployment is described, it is essential to present the key third-party libraries that were added to develop the main features of the project.

5.2.1 React Router DOM

As stated in section 4.5, the react-router-dom library allows the navigation between different "pages" within a SPA. In addition to changing the URL inside the browser address bar, it also allows to bookmark pages, use session history and programmatically navigate the user to a specific path.

In this project, the main use of this library is to ensure the navigation between the three defined paths: /login, /signup, and /main. The routing library together with the authentication token, allows the user to navigate in the application taking into consideration his authentication status.

If the user is authenticated, the SPA redirects to the /main page. If the user is not authenticated, the system navigates between the /login or /signup page, depending on the URL path. The Redirect component ensures that the user is redirected to an existing page if the URL path doesn't exist. The authentication logic will be explained ahead.

5.2.2 React Hook Form

The react-hook-form is a library that helps to create and validate forms in React. React Hook Form adopts the use of uncontrolled inputs using references instead of depending on the state to control the inputs. This approach makes the forms more performant and reduces the number of re-renders.

React Hook Form also supports schema-based form validation where it is possible to pass the schema to useForm as an optional config. React Hook Form will validate the input data against the schema and return with either errors or a valid result. To ensure the schema-based form validation, another third-party library name Yup was installed.

In this project, every form is developed using the React Hook Form library, together with the Yup schema validator and Material-UI components.

5.2.3 Material-UI

Material-UI is a React component library based on Google Material Design that allows a faster and easier stylized web development. In order to speed up the development process, this library was. Most parts of the application UI components such as text fields, buttons, icons, tables, inputs, selects, modals, forms, and sidebar are imported from the Material-UI library.

The fact that the library is well documented, the components have a visual consistency, and the ability to be edited individually, not relying on style-sheets, makes this library a great choice for improving the application frontend.

5.2.4 Express

Express is a Node.js web application framework that offers a comprehensive range of functionality for both web and mobile apps. Using a variety of HTTP utility methods and middleware, it is quick and easy to build a powerful API [75].

In this project, Express.js is used because it makes building the server-side logic with Node.js much easier. The middleware-focused approach allows the developer to produce middlewares, or implement third-party library middlewares, like bodyParser or express validator, which simplifies things and allows the developer to focus on the core business logic of the server application instead of reinventing the wheel.

Express Validator is an Express middleware library that was incorporated in server-side data validation.

Body-parser is the Node.js body parsing middleware. It is responsible for parsing the incoming request bodies in a middleware before the middlewares created by the user handles them.

5.2.5 Mongoose

Mongoose is an Object Data Modeling (ODM) library for MongoDB and Node.js. It manages relationships between data, provides schema validation, and is used to translate between objects in code and the representation of those objects in MongoDB. In simple words, Mongoose acts as an intermediate between MongoDB and Node.js.

In this project, Mongoose is used to connect the backend server to the database, model the application data, validation and query building, strengthening the server-side logic development.

5.2.6 Bcryptjs

Bcryptjs is a secure way to store passwords in a database. This library is used to hash the user's passwords in order to secure the user information. To hash the password bcryptjs use the salted hashing method, which generates random bytes (the salt) and combines them with the password before hashing, creating unique hashes across each user's password.

5.2.7 Jsonwebtoken

JSON Web Token (JWT) is an open standard that defines a compact and self-contained way for securely transmitting information between parties as a JSON object. This information can be verified and trusted because it is digitally signed. JWT can be signed using a secret (with the HMAC - Hash-based Message Authentication Code - algorithm) or a public/private key pair using RSA - Rivest-Shamir-Adleman - or ECDSA - Elliptic Curve Digital Signature Algorithm [76].

In this project, the JWT is implemented for authorization, section 5.3.1. Once the user is logged in, each subsequent request will include the JWT generated for that user, allowing the user to access routes, services, and resources that are permitted with that token.

5.2.8 Web Push

Web push messages require that push messages triggered from a backend be done via the Web Push Protocol and, if data is sent with the push message, the information must be encrypted according to the Message Encryption for Web Push spec. The web-push library makes it easy to send push notifications messages. In this project, this library was used to simplify the push notifications implementation that is described in section 5.3.8.

5.3 Main Features

In the following subsections the main developed features are presented but, first, it is important to make some considerations:

- Each sequence diagram presented shows the success cases.
- All the features explained next, except Login and Signup, assume that user is authenticated and on the /main page.
- In every request, except for Login and Signup requests, the authentication token is sent as an header with the user id and role encoded.
- In all requests for which the user id is sent as the request body, the backend does a double check, verifying that the user id matches the user id encoded in the token.

5.3.1 Authentication and Authorization

Authentication is crucial in a web application because access to certain actions or pages can be restricted. Authorization is the process of restricting user access via specified roles and privileges, whereas authentication is the process of proving an individual's identity.

In most web applications, the user gets authenticated after logging in to the app, having access to specific resources. The user authentication state is usually maintained by a session or a JWT.

In this project, since the frontend is a single-page application and the backend is a REST API, it is impossible to create and store sessions in the backend server. The backend is stateless and decoupled from the frontend. To solve this, instead of creating and saving sessions, the backend generates a token that is returned to the front end, where it is stored as cookie or in the local storage. The token is then attached to all frontend requests that request a protected resource. Lastly, the backend validates the token. This logic is shown in the Figure [5.3](#).

As stated before, this token is generated with the help of the JWT library. When creating the token, using the function `jwt.sign()`, it is possible to encode any data and it is necessary to define a private key. This private key, which is only known by the backend server, is what allows the backend to validate the token that is attached to a frontend request. This means that, if any user tries to fakes the token, they will generate an invalid token because they don't know the private key.

In this project, the token is created when the user logs in to the application. The frontend sends a request, with the user information, the backend checks if this information is valid and generates a token. When generating this token, the backend encodes some data user: username, name, email, and role. This information will be available in every frontend request, where the token is attached to.

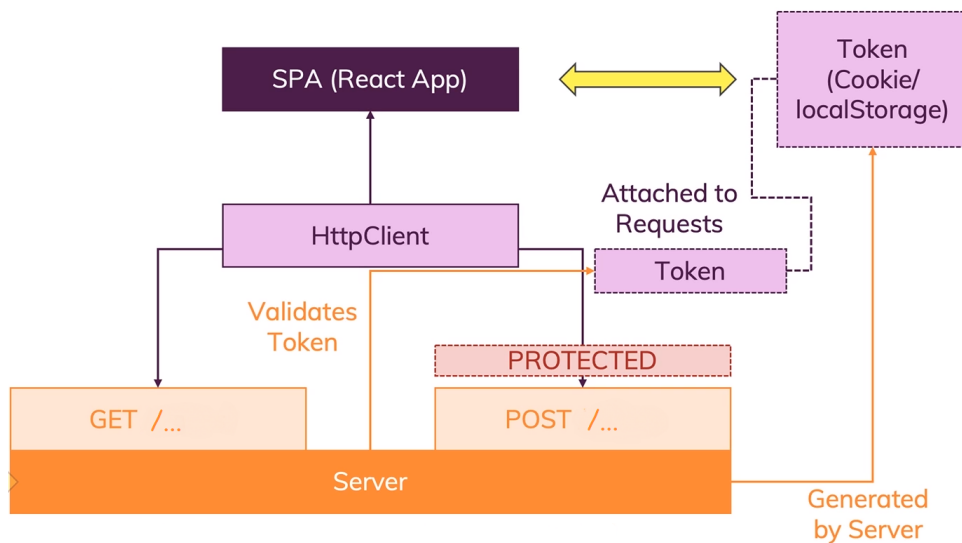


Figure 5.3: Authentication
[74]

The backend has the power to choose which routes to protect or open to any user. This protection can simply be implemented in a middleware that checks if the user is authenticated. This leads to the other presented concept, Authorization.

In this project, authorization is implemented in two ways. The first way is to implement a middleware, which runs before the others and checks if the user is authenticated. Every request except login and signup checks if the user is authenticated. The second way is, if the user is authenticated, in specific requests, such as adding a component, the user's role, previously encrypted, is decrypted, and it is verified. If the user's role matches the role required to perform this function, the request proceeds, if not, it is canceled and an error message is displayed.

On the front-end, it is also possible to include the second type of authorization. When the user is logged in, the frontend not only receives the token but also receives the user's name, username, and role. Thus, there are React components, such as the button to add a component, which may or may not be enabled, depending on the user's role.

5.3.2 Data Validation

The data validation takes place in the 3-tier architecture (frontend, backend, database).

On the frontend, the validation of the information entered by the user takes place thanks to the React Hook Form and Yup libraries. It is possible to validate if the email entered is an email, if the password entered has at least 5 characters, etc.

In the backend, after this input information is sent to the API endpoint, the information is also validated. The express validator library allows you to validate the information that arrives at the controller. It is possible to check if a field is not empty or if it has a minimum length of 5, for example.

Validation of information for the database is achieved through the mongoose library. When defining database schemas with this library, it is possible to define which field has a certain property. For example, it is possible to establish that the password must have at least 5 characters.

This validation in the three parts of the project guarantees data integrity throughout the entire project. The data validation occurs in every feature that will be present next.

5.3.3 User Management

In the following subsections, the features related to the User Management will be described: Login, Signup and Edit User Information.

5.3.3.1 Login

When the user accesses the /login page, the SPA returns the Login component and its subcomponents, as shown in Figure 5.4.

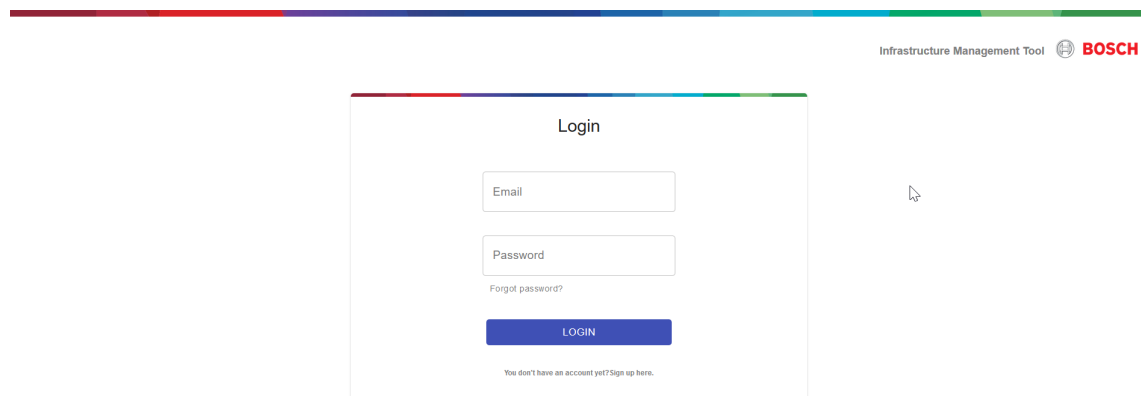


Figure 5.4: Login UI

The user fills in the form fields with his/her credentials and clicks the login button. Immediately, the frontend sends an HTTP request to the backend, taking into account the previously defined API endpoints. In this case, it is a POST request to the endpoint /users/login with the email and password values entered by the user. The backend is responsible for doing all the routing, taking

into account the path defined in the endpoint. When the information arrives at the controller in JSON format, the controller, in this case, sends a query to the database, to obtain all the data from the user, taking into account his email address, which is unique. The database responds with this data. The controller, in turn, decrypts the password sent by the database, using the bcrypt library, and compares it with the password entered by the user. If everything is ok, the controller, through the JWT library, creates the authentication token. This token has encoded the user id and role. The controller sends the token, user id, username, user's username, and role as the response. The frontend receives this information and automatically logs the user into the system and stores the token in the browser's local storage, to be accessible for next time, until it expires. SPA redirects the application to the /main page once the user is authenticated. The sequence diagram corresponding to this description is in Figure 5.5.

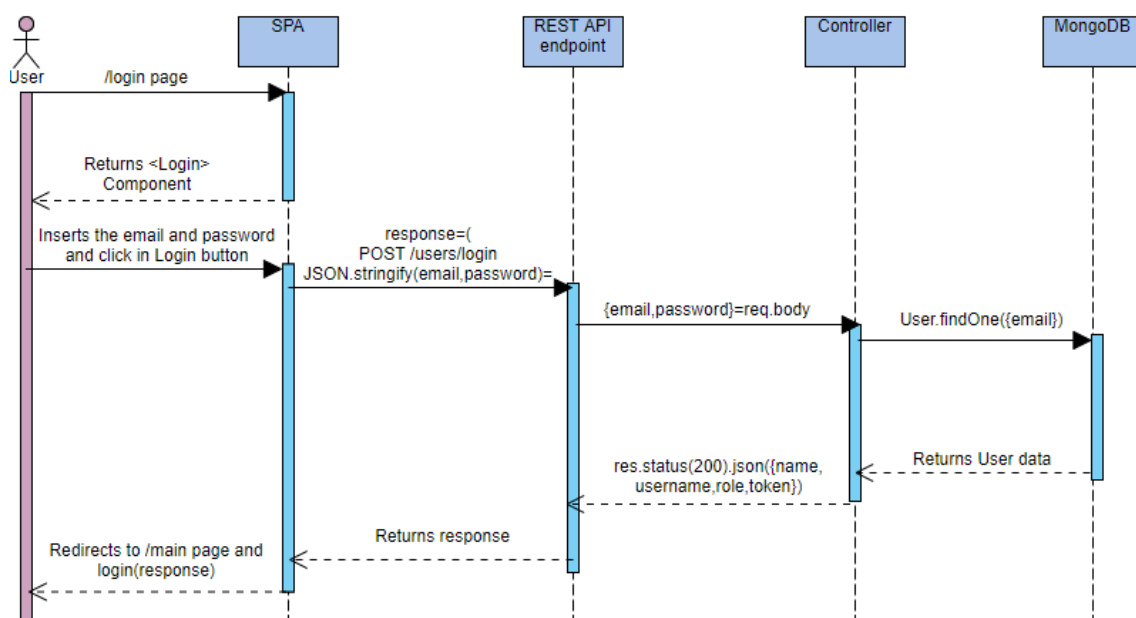


Figure 5.5: Login Sequence Diagram

5.3.3.2 Signup

The user accesses the /signup page and the application shows the <Signup> component and its subcomponents, as shown in Figure 5.6.

The user fills in the form with name, username, email, and password and clicks on the button to register. Then the frontend sends a POST request to the API endpoint /users/signup with the information entered as the request body. The backend, taking into account the endpoint's path, routes, and accesses the controller signup. The controller, in turn, requests the database looking for users with the email or username received. If it doesn't find any user, the controller sends another query to the database where the purpose is to store the user's information. This information contains the name, username, email, and password entered by the user and also the team, section,

Infrastructure Management Tool **BOSCH**

Signup

Name

Username

Email

Password

SIGN UP

You already have an account?

Figure 5.6: Signup UI

and role assigned by default to a new user. The role of this new user is Basic, as defined in the project requirements. The controller confirms the success of the operation by sending the HTTP status code 200, and the frontend, taking into account that there was no error, redirects the UI to the /login view. The sequence diagram described above is visible in Figure 5.7:

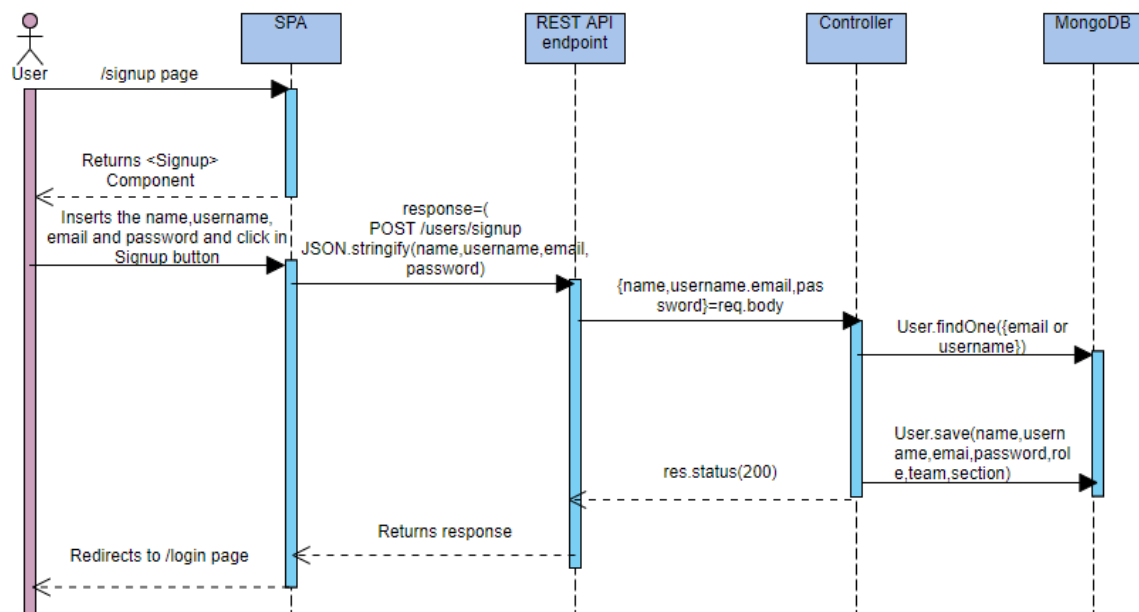


Figure 5.7: Signup Sequence Diagram

5.3.3.3 Edit User Information

The user, to change his/her user information, clicks on the user icon available on the /main page. The frontend sends a POST request to the endpoint /users/user with the user id as the body to get

the user data. The application renders the UI with its components, as visible in Figure 5.8.

Figure 5.8: Edit User Information UI

The user fills in the form with the new username, password and confirms the password. Then clicks on the Edit button. The frontend sends a PATCH request to the endpoint `/users/edit` with the information entered by the user and also his id. The controller checks if the user id matches the id encoded in the token attached to the request. After, the controller sends a query to the database which checks if there is any user with the new username desired by the user. In case it doesn't exist, the controller sends another query to the database where it updates the user data to the new username and password. Given the success of the operation, the controller sends a successful response and the username to update the UI header. The frontend shows a success notification. This sequence diagram is shown in Figure 5.9:

When a user wants to edit other users' information, the endpoint is the same. The main difference is in the controller logic.

When a Scrum Master tries to change a user's information, the controller takes the id and role that is encoded in the token which is attached to the request. Through the id, it asks the database for this user's team and checks if it is the same as the user whose information will be changed. Finally, check if the role is Scrum Master. If confirmed, the user information can be edited.

When an Admin tries to change a user's information, the controller checks if the role that is encoded in the token is Admin. If it is, the user information can be edited.

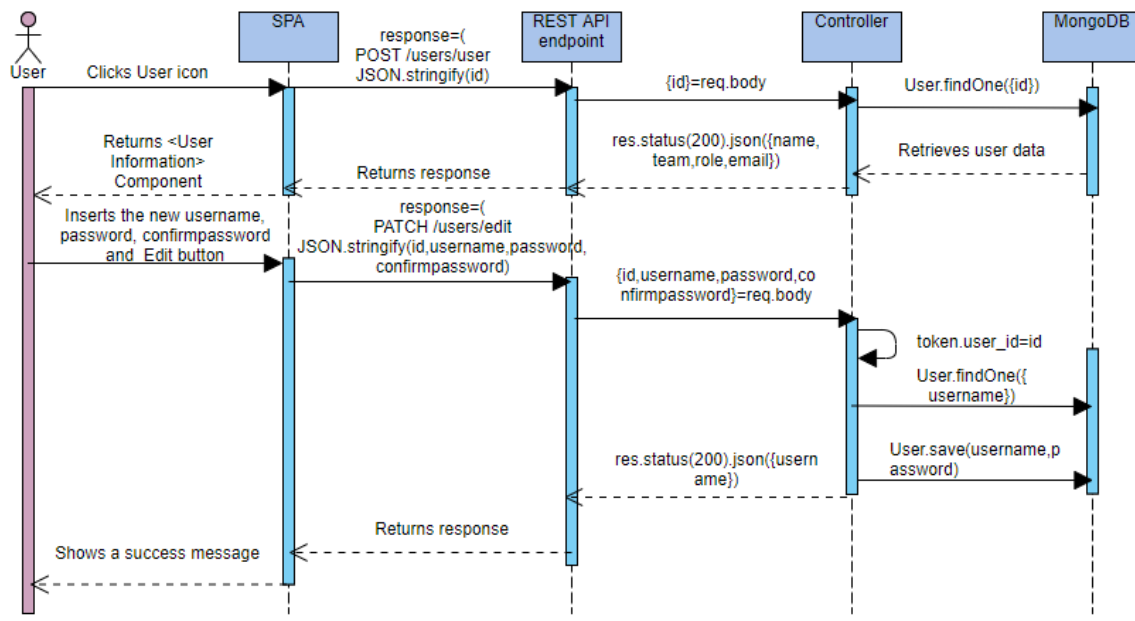


Figure 5.9: Edit User Information Sequence Diagram

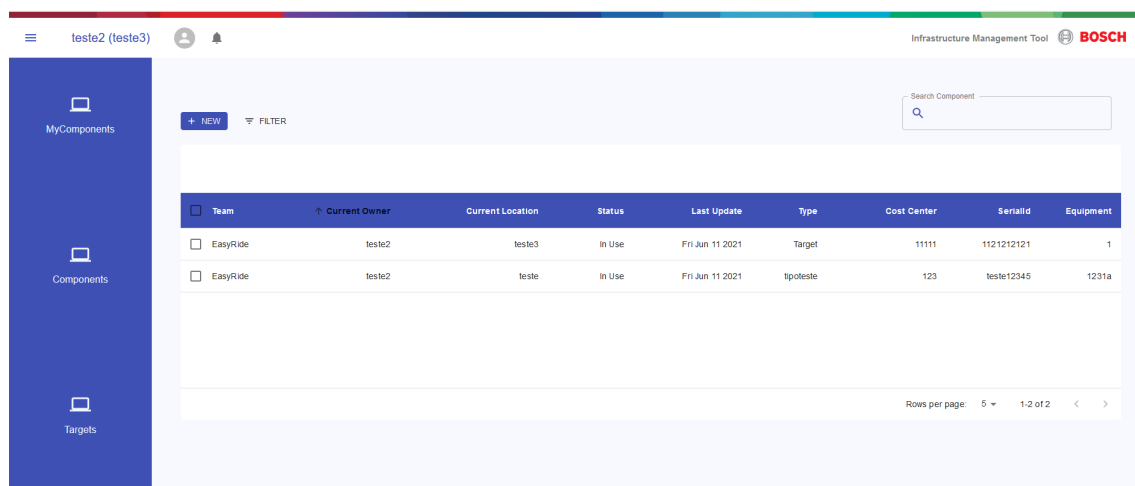
5.3.4 Components and Targets Management

In the following subsections the operations related to the management of Targets and Components will be presented: Add, Edit and Delete, but before that, it is important to make some considerations:

- Before each feature, it is assumed that the user is properly authenticated and on the /main page. The user must also be in the Components catalog or Targets Catalog view, shown in Figure 5.10.
- The sequence diagrams presented for each feature are for an Admin user because they are the most general use cases. The specifics for each role are explained in the text.
- Lastly, the features presented are all related to components only. Targets are a specific type of component (type=target). Thus, although there are a series of endpoints related to targets, the only difference is that the target is a component whose type=target. There is no need to extend the content of this chapter and the others to this specificity.

5.3.4.1 Add

Adding a component is a task that can only be performed by an Admin or a Scrum Master if the component team is the same as the Scrum Master team. This is ensured by the system both in the frontend and backend, by not displaying the button in the UI for a Basic user and verifying the user role in the backend, respectively. Both the Admin and the Scrum Master can assign the component to some other user, provided that, in the case of the Scrum Master, the user and the



The screenshot shows a web application interface for an 'Infrastructure Management Tool' by Bosch. On the left is a blue sidebar with navigation links: 'MyComponents', 'Components', and 'Targets'. The main area displays a table of components. Above the table are buttons for '+ NEW' and 'FILTER', and a search bar labeled 'Search Component'. The table has columns: Team, Current Owner, Current Location, Status, Last Update, Type, Cost Center, SerialId, and Equipment. There are two rows of data.

Team	Current Owner	Current Location	Status	Last Update	Type	Cost Center	SerialId	Equipment
EasyRide	teste2	teste3	In Use	Fri Jun 11 2021	Target	11111	1121212121	1
EasyRide	teste2	teste	In Use	Fri Jun 11 2021	tpoteste	123	teste12345	1231a

At the bottom right of the table area, it says 'Rows per page: 5' and '1-2 of 2'.

Figure 5.10: Example Catalog UI

Scrum Master are from the same team. When a component is added, a dueDate is automatically set to two weeks after.

A component can be added with two statuses: available or in use. In the first case, the component never has a user assigned, while in the second case, it must.

When the user clicks the button to add a component, the application returns the React component `<Add Component>`. The user fills in the form correctly and clicks the Add button. The frontend immediately sends a POST request to the endpoint `/components/add` with the id of the user adding the component and the component information (team, user, location, status, date, type, cost, serialId, equipment). After receiving this data, the controller promptly checks if the role that is encoded in the token attached to the request is equal to admin. If everything is ok, the controller sends queries to the database for a series of checks: check if the serial ID of the component already exists and if the user, if assigned, also exists. If everything is good, the controller sends a query to the database to save the component information and another query to the database to add a new row in the History table (with, for example, the id of the user who added the component and the component id as parameters). The controller sends a success response, which the frontend then translates into a success message that appears in the UI. The described sequence diagram is shown in Figure 5.11.

In the case of the Scrum Master, the process is very similar. When the controller verifies that the user who is trying to add a component has the role of Scrum Master and not the admin, it makes further checks. The controller quickly sees if the team the component is associated with is the same as the Scrum Master, and if a user is assigned, it also checks if the team is the same. In case these two conditions are verified, the process is similar to that of the Admin.

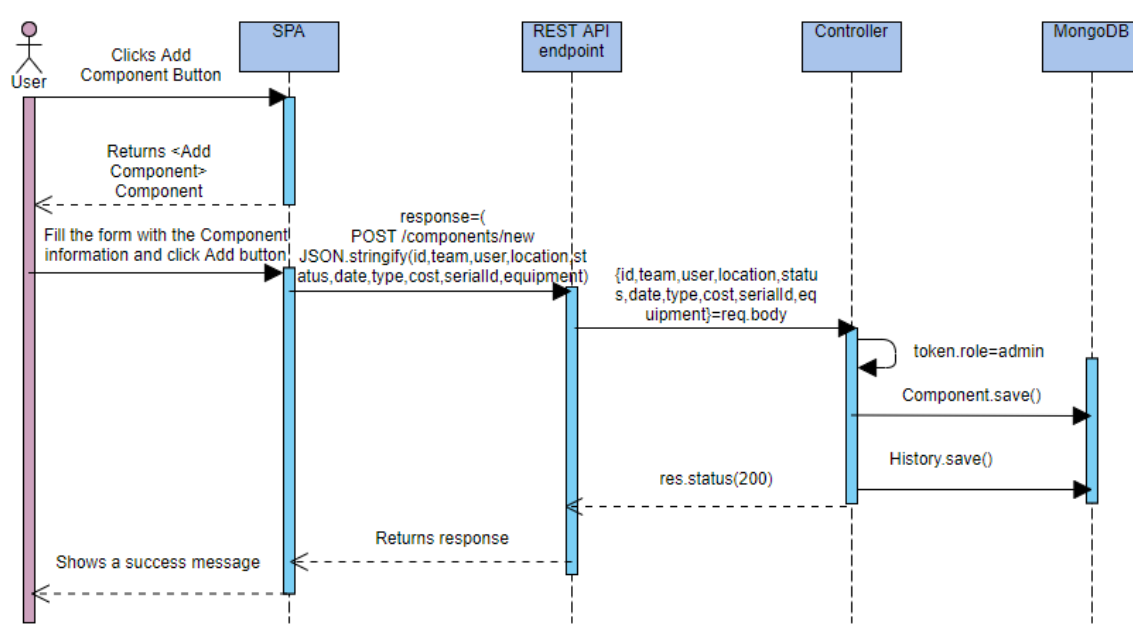


Figure 5.11: Add Component Sequence Diagram

5.3.4.2 Edit

Editing a component is a task that any user can perform, but with certain conditions, taking into account the user's role. In addition to the backend verifying these conditions, the frontend also makes a point of altering the UI, disabling the edit button, for example, taking into account the user's role.

A Basic user can edit a component if assigned, component user attribute = user id, or a component whose status is available and the team is the same as the Basic user.

A Scrum Master can edit a component that is assigned to him and can also assign a component, whose status is available, to himself or to a user of his/her team.

An Admin can edit any component. Figure 5.12 shows the Edit Component sequence diagram for an Admin user.

The user clicks the edit button to edit the component. The frontend makes a POST request to the backend, sending the component id as the message body, expecting to receive the component's information in response. After the controller requests this information from the database and sends it back to the frontend, it loads a form with the component information.

After, the user edits the information and clicks the edit button. The frontend makes a PATCH request to the endpoint /components/edit with the component id, id of the user who wants to make the change, and the other attributes of the component. The controller promptly checks if the role

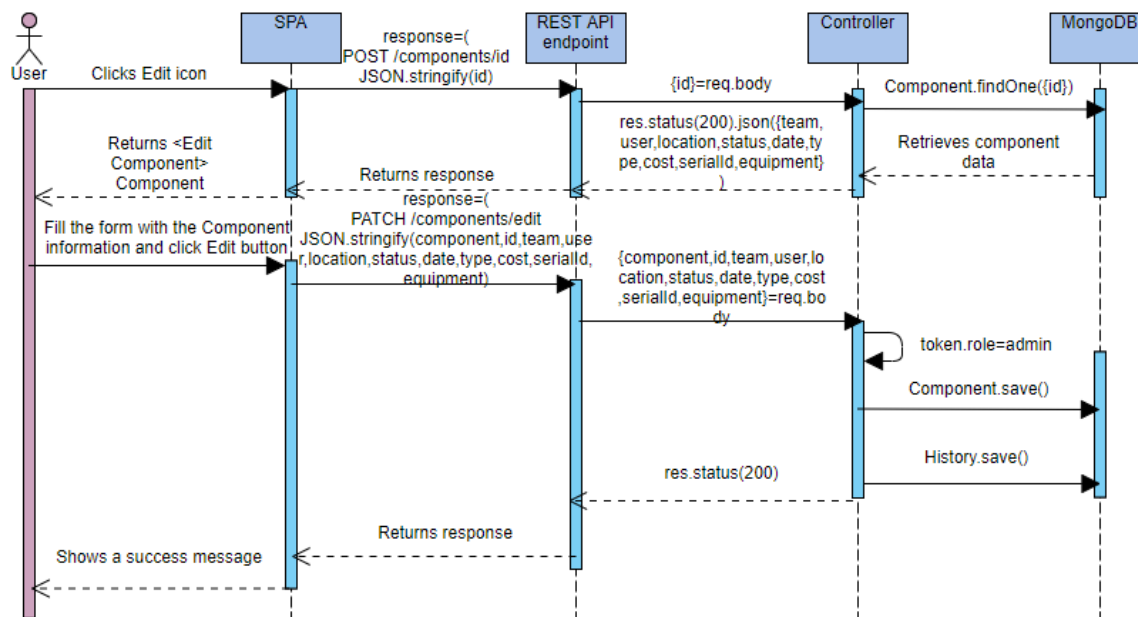


Figure 5.12: Edit Component Sequence Diagram

that is encoded in the token attached to the request is equal to admin. If so and after a series of validations, such as checking if the serial ID already exists, the controller sends a query to the database in order to update the component. The controller also sends a query to add a new row in the History table with the component id, id of the user who edited the component, date, and a brief description. If the component status was in use and moved to available, the end date attribute is also added. The frontend after receiving the success response from the backend shows an "Edit Successful" notification.

The process of editing a component for a user whose role is Scrum Master is similar. The only conditions that are also validated by the controller are: the component's team must be the same as the Scrum Master's team, and, if a user is assigned, this user's team must be the same as the Scrum Master's team. In the case of the Basic user, the component team must be the same as the user, and if the component is in use, the user is the user.

5.3.4.3 Delete

The Delete Component function is very similar to the Add Component feature. The Admin can delete any component while the Scrum Master can only delete components that belong to his team. The component's status must be available.

The workflow to delete a component starts with the user confirming that they want to delete the component after the application has sent a confirmation dialog. The frontend, after the user's decision, sends a DELETE request to the endpoint `/components/delete` with the component id and user id. The controller verifies if the role that is encoded in the token attached to the request is

equal to admin and if the status of the component is available. If so, the controller sends a query to the database to delete the component. The controller then sends a query to the database to update the description attribute to "This component no longer exists" of all documents in the History table that exist for this component. After the controller sends a successful response, the frontend shows a notification that the component has been deleted. The described sequence diagram is shown in Figure 5.13.

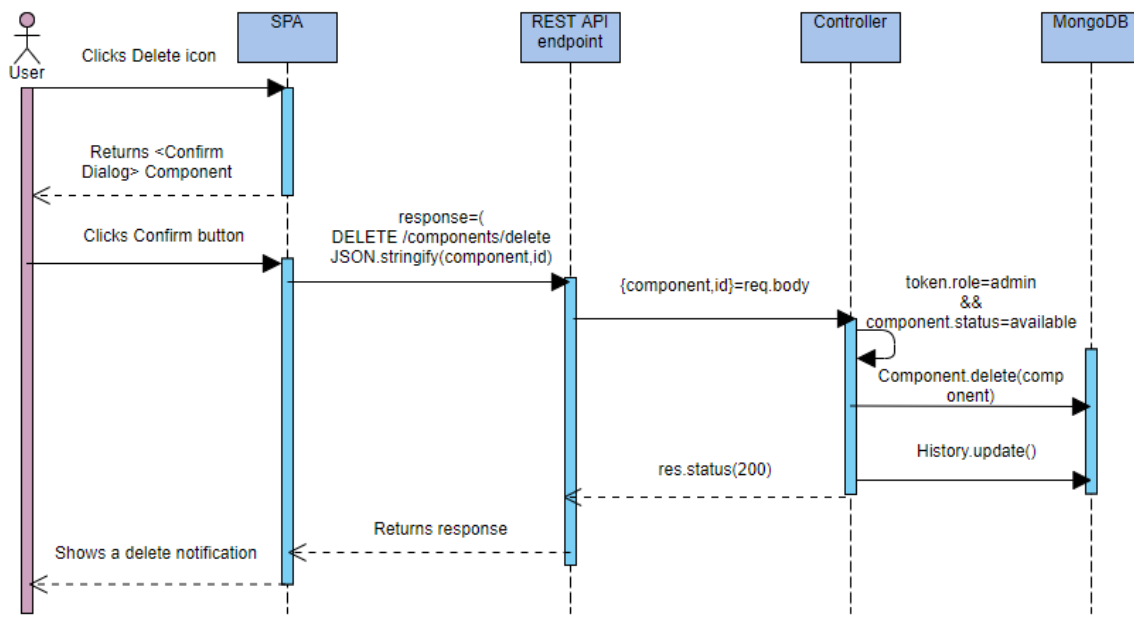


Figure 5.13: Delete Component Sequence Diagram

In the case of the Scrum Master, the controller also validates that the component team is the same as the Scrum Master.

5.3.5 Cables Management

In the following subsections the operations related to the management of Cables will be presented: Add, Edit and Delete taking into consideration the user's role. Like in the Components and Target Management section, the sequence diagrams presented for each feature are for an Admin user while the specifics for each role are explained in the text. It is also assumed that the client is in the Cables Catalog view.

5.3.5.1 Add

Add Cables is a task that can be performed by all users. An Admin user can add any type of cables while a Basic or Scrum Master user can only add cables whose section is his.

To add cables, the user clicks on the "Add Cables" button. The frontend returns the '<Add Cables> React component'. The user fills in the form with the information of the cables: section,

location, description, total quantity, available quantity, quantity in use, and clicks on the add button. The frontend sends a POST request to the endpoint `/cables/new` with the data entered by the user. The controller checks if the role that is encoded in the token, that is attached to the request, is equal to admin. It then sends a query to the database to store the cables information. The backend sends a successful response to the frontend, which in turn displays a success notification. The equality of $\text{total quantity} = \text{quantity in use} + \text{quantity available}$ is verified in both the frontend and the backend. The described sequence diagram is shown in Figure 5.14.

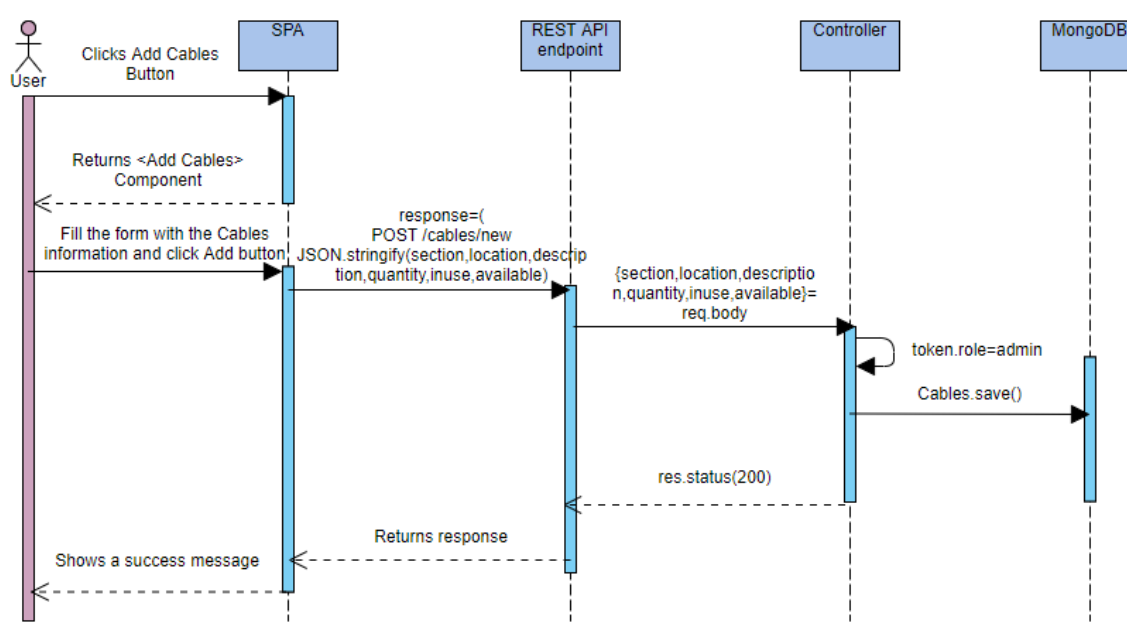


Figure 5.14: Add Cables Sequence Diagram

In case the user is a Basic or Scrum Master user, the controller, after verifying that the user is not an Admin, checks if the section of the cables that the user wants to add is equivalent to the user section.

5.3.5.2 Edit

Edit Cables is a task that can be performed by all users. An Admin user can edit any set of cables while a Basic or Scrum Master user can only edit cables whose section is his.

The user clicks the edit button to edit the cables. The frontend makes a POST request to the backend, sending the cables id as the message body, expecting to receive the cables' information in response. After the edit cables controller requests this information from the database and sends it back to the frontend, it loads a form with the cables information.

After, the user edits the information and clicks the edit button. The frontend makes a PATCH request to the endpoint `/cables/edit` with the cables id and the other attributes of the cables. The

edit cables controller promptly checks if the role that is encoded in the token attached to the request is equal to admin. If so, the controller sends a query to the database in order to update the cables. The frontend after receiving the success response from the backend shows an "Edit Successful" notification. The equality of total quantity = quantity in use + quantity available is verified in both the frontend and the backend. The described sequence diagram is shown in Figure 5.15.

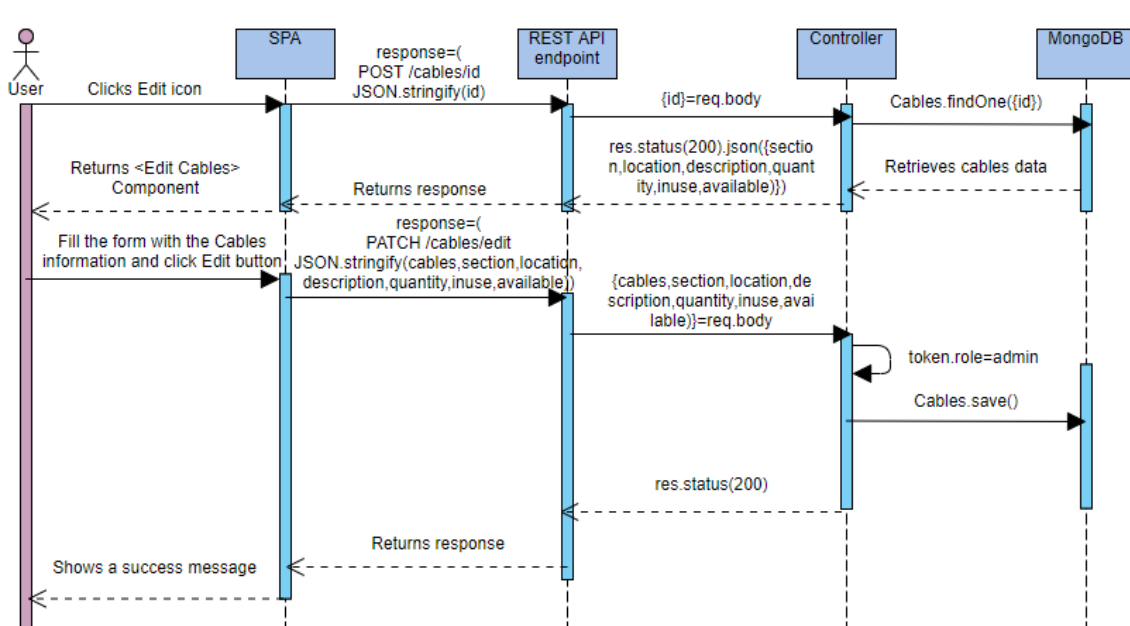


Figure 5.15: Edit Cables Sequence Diagram

In case the user is a Basic or Scrum Master user, the controller, after verifying that the user is not an Admin, checks if the section of the cables that the user wants to edit is equivalent to the user section.

5.3.5.3 Delete

The Admin can delete any set of cables while the Scrum Master can only delete cables that belong to his section. A Basic user can't delete cables.

The sequence to delete a set of cables starts with the user confirming that he/she wants to delete the component after the application has sent a confirmation dialog. The frontend, after the user's decision, sends a DELETE request to the endpoint /cables/delete with the set of cables id. The delete cables controller verifies if the role that is encoded in the token attached to the request is equal to admin. If so, the delete cables controller sends a query to the database to delete the cables. After the controller sends a successful response, the frontend shows a notification that the set of cables has been deleted. The described sequence diagram is shown in Figure 5.16.

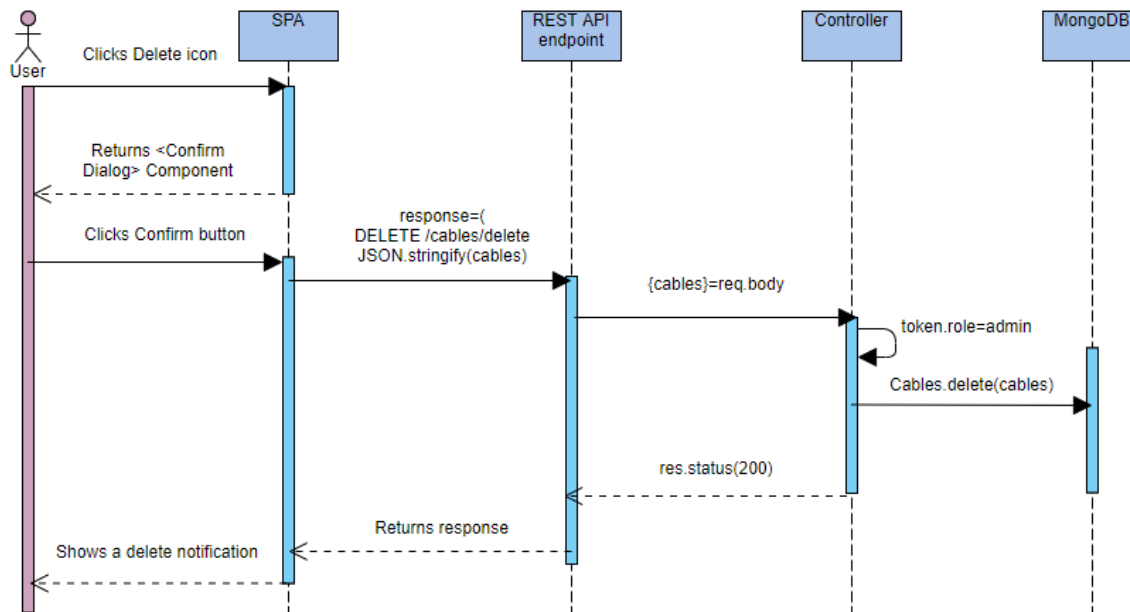


Figure 5.16: Delete Cables Sequence Diagram

In the case of the Scrum Master, the controller also validates that the cables section is the same as the Scrum Master.

5.3.6 History

In the following subsections, the features related to the History requirement will be described: Components History and User Components History.

5.3.6.1 Components History

Any user who is authenticated can view the history of any component, Figure 5.17. To do this, the user just selects one or more components and click on the "View More Info" button.

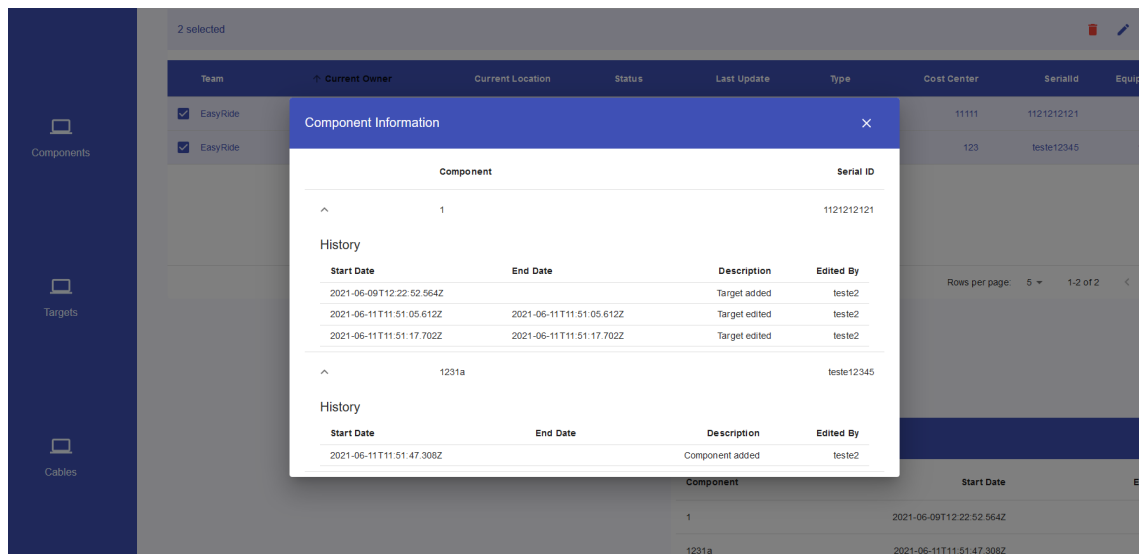


Figure 5.17: Component History UI

The frontend sends a POST request to the backend with the ids of the selected components. The controller sends a query to the database with the expectation of receiving all documents from the History table whose component attribute ids are equal to the ids received in the request. The controller sends in response an array containing for each element the component, description, the end date, the start date, and who edited the component. The frontend renders the React component `<Components History>` which contains the data sent by the backend. The described sequence diagram is shown in Figure 5.18.

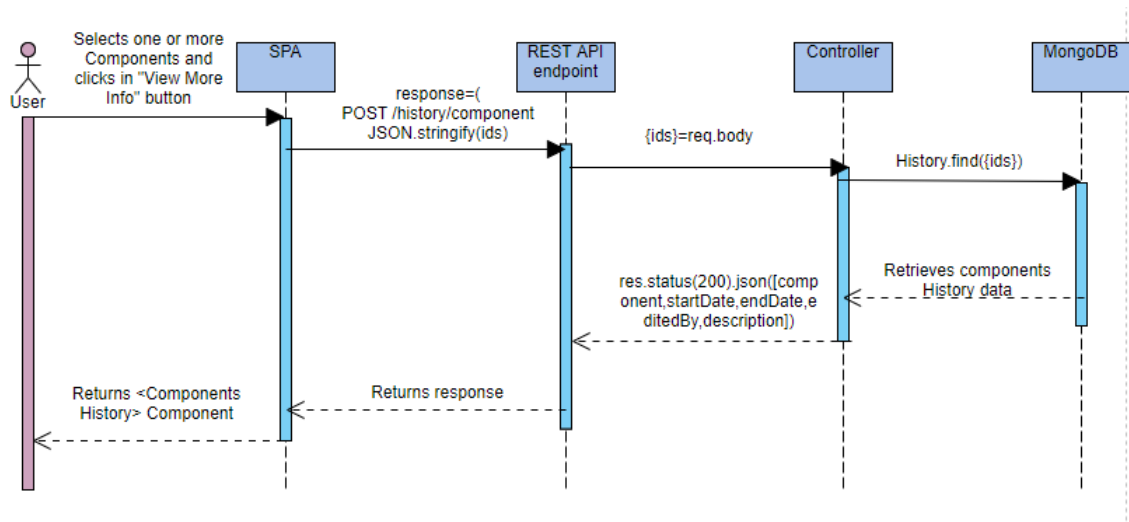


Figure 5.18: Components History Sequence Diagram

5.3.6.2 User Components History

Any user who is authenticated can view the history of the components that belonged to him. To do this, the user just opens the MyHistory menu, Figure 5.19.

MyHistory		
Component	Start Date	End Date
1	2021-06-09T12:22:52.564Z	
1231a	2021-06-11T11:51:47.308Z	
60c34e74bbb38f355cffe487	2021-06-11T11:52:20.104Z	2021-06-11T13:41:39.661Z

Figure 5.19: User History UI

The frontend sends a POST request to the backend endpoint `/history/user` with the id of the user. The controller sends a query to the database with the expectation of receiving all documents from the History table whose user attribute id is equal to the user id received in the request. The controller sends in response an array containing for each element the component, the end date and the start date. The frontend renders the React component `<User Components History>` which contains the data sent by the backend. The described sequence diagram is shown in Figure 5.20.

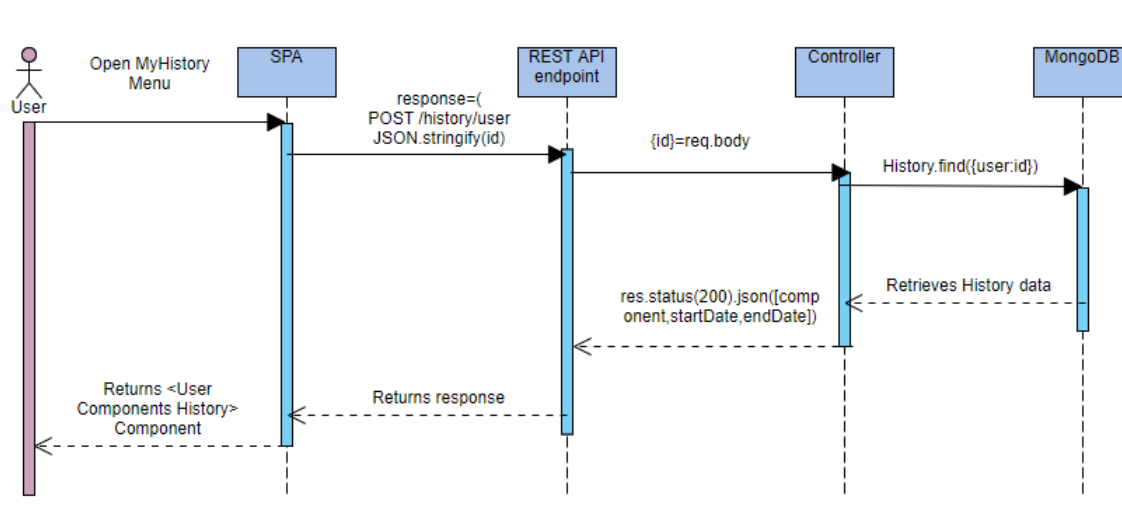


Figure 5.20: User Components History Sequence Diagram

5.3.7 Search and Catalog

The Search and Catalog feature consists of a dynamic table with pagination and ordering/sorting by columns, a search bar, and filters. In the project, there are 4 types of catalogs: MyComponents, Components, Targets, and Cables. Although there are different catalogs, the Components catalog will be used to exemplify all the logic behind this feature which is similar to all catalogs.

Team	Current Owner	Current Location	Status	Last Update	Type	Cost Center	SerialId	Equipment
STICCMEN	admin	Locker Teste	In Use	Wed May 19 2021	Teste	100	1	Teste 1
EasyRide	admin	Locker	In Use	Wed May 19 2021	Teste	teste	teste1	teste
EasyRide	Teste	Locker	In Use	Wed May 19 2021	Teste	teste	teste2	teste2
STICCMEN	sico	Locker Teste	In Use	Mon May 31 2021	Teste	200	2	Teste 2
EasyRide	sico	asa	In Use	Mon May 31 2021	asas	asasa	sasa	asasasa

Rows per page: 5 1-5 of 7

Figure 5.21: Extended View UI

The UI of this feature, as seen in Figure 5.21, consists of a table, a search bar, and a menu to apply filters. When the user clicks on the Components page, to view their catalog, the frontend sends the first request to the backend to obtain the components data stored in the database to fill the rows of the table. This request takes into account the selected filters, the page of the table, the number of rows per page, if any word is written in the search bar, if any column is selected to sort, and in what order. Whenever some of these variables change, a new order is sent to the backend. The sequence diagram that describes this operation is visible in Figure 5.22.

When the controller receives these data, it only sends a query to the database with all the restrictions defined. The query indicates that it wants only the components whose team, type, status, or cost match the values of the respective filters and that the equipment or serialID match what is written in the search bar, if something is written. Before answering with the components, the query also defines that they are ordered taking into account what is defined in the order and orderBy variables. Lastly, the number of components is limited by the page and the number of rowsPerPage chosen by the user in the frontend table. As a response from the backend, components and their information are sent, as well as the total number of records. The table in the frontend is filled with this information.

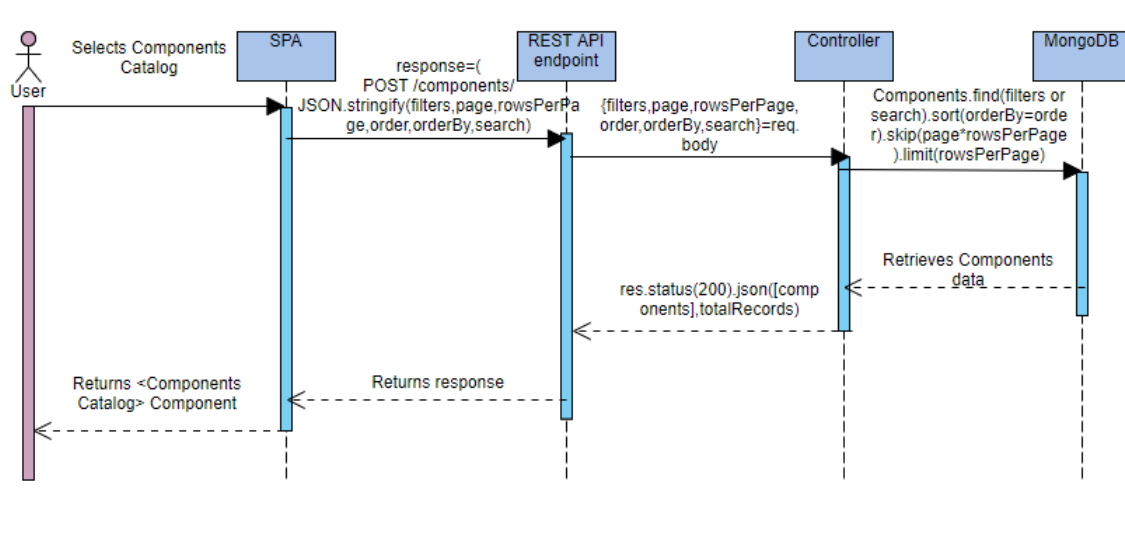


Figure 5.22: Components Catalog Sequence Diagram

The existing filters are defined through a single request that happens when the user clicks on the components catalogs page. This request will fetch all existing teams and sections from the teams and sections database tables, and also search in the components table for the different values of type and cost. This process is similar in the others catalogs, with the necessary adaptations.

5.3.8 Notification System

The notification system developed in this project was implemented in two ways: via push notifications and built-in on the tool. In the next subsections, these implementations are described as well as the notification types and the notifications management.

5.3.8.1 Notification Types

As described in chapter 4, there are 3 types of notifications:

- **Check Status:** The notification is triggered when the user is using the component for more than 2 months.
- **Estimated Time:** The notification is triggered when the estimated use time is reaching the end.
- **Over Time:** The notification is triggered when the estimated use time expired and he also sent to the upper-level roles (Scrum Master and Admin).

To implement these notifications in the system, code was added to the Notifications controller. This code is scheduled and only runs few times a day depending on the notification type:

- **Check Status:** 1 time a day.

- Estimated Time: 2 times a day.
- Over Time: 4 times a day.

The code is independent for each type of notification and it doesn't need to be always running because it essentially checks the estimated end date (dueDate) of a component's use.

The Check Status type notification checks two attributes of the Component class: dueDate and Ack. The latter refers to the date the component was last updated. If the dueDate has not passed yet and if the last Ack was more than two months ago, the backend, for each component, sends a query to the database to create a checkstatus notification. As an attribute of this notification, the user associated with the component is also included.

The Estimated Time type notification checks if a component's dueDate is within the next seven days. If so, the backend, for each component, sends a query to the database to create an estimat-edtime notification. As an attribute of this notification, the user associated with the component is also included.

The Over Time notification is created when the component's dueDate has already passed. The backend not only asks the database to create a notification for the user associated with the component, but it also asks to create for the Scrum Master of the component/user team and the Admin.

When any component is edited, becoming available, or deleted, the respective notifications are also deleted.

5.3.8.2 Push Notification System

Push notifications are very common in browsers and mobile apps. This type of notification system is implemented in the project to ensure that the user receives the notifications even if the tool is not open on the browser. Before describing how this system is implemented in the project, it is necessary to understand how the push notification cycle works:

1. When the user opens the app for the first time, a service worker is installed in the browser. The service worker will allow the notifications to be shown even if the application is closed.
2. A user receives a request to display notifications.
3. After receiving permission to display notifications, the browser hands credentials and other service information to the application. This data must be sent to the backend and stored in the database.
4. Using the previously received credentials, the backend requests the service provider, which in turn sends a notification to the service worker.
5. The service worker receives a notification and shows it.

To implement this system, code was added in the backend and frontend. The backend, with the help of the web-push library, is responsible for generating the VAPID keys, which will serve as an authentication measure to send notifications via the push service, and for sending the notifications. The frontend is responsible for installing the service worker in the browser, taking into account the VAPID - Voluntary Application Server Identity - keys generated by the backend, and asking the user if he wants to subscribe to the notifications. If yes, the frontend sends to the backend the necessary credentials to be able to send push notifications to that browser (Google Chrome differs from Firefox for example). The backend sends the notifications through the web-push library and has as parameters the credentials received from the frontend and the notification payload (title, body, etc).

In the project, notifications are sent through the backend whenever the scheduled code runs and the necessary conditions are met. This means that whenever a notification is created in the database, it is also sent to the user by the push service.

5.3.8.3 Tool Notification System

Unlike the push notifications system, the notification system built in the tool does not display the notification as soon as it is sent by the backend. The notifications can be found in a menu, accessible to the user, in which he can make decisions, Figure 5.23.

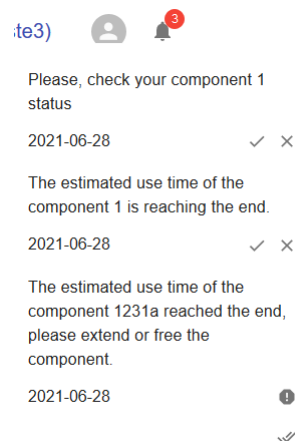


Figure 5.23: Notifications Menu UI

The user clicks on the notifications icon and the frontend immediately sends a POST request to the endpoint `/notifications/user`. The controller, in turn, through the user id sent with the request or through the user id that is encoded in the token attached to the request, sends a query to the database requesting the notifications that are associated with that user. When the frontend receives this information, it displays the React `<Notifications Menu>` component, which contains the notifications related to that user. When an action is taken on a notification, this request is repeated. The sequence diagram that describes this operation is visible in Figure 5.24.

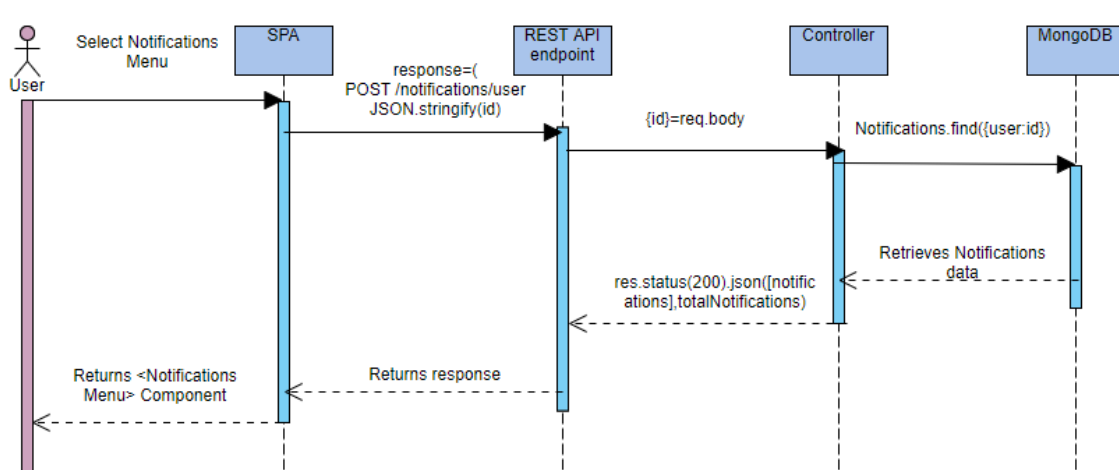


Figure 5.24: Notifications Menu Sequence Diagram

5.3.8.4 Notifications Management

For each type of notification, different actions can be taken: acknowledge or clear a notification. It is also possible to acknowledge all Check Status and Estimated Time notifications simultaneously.

In the push notifications, although all types of notifications are sent, it is only possible to take individual actions on the Check Status and Estimated Time notifications. In the tool notifications, it is possible to take all the actions available. Thus, the sequence diagrams described below are based on the actions taken in the tool. All backend logic is similar for actions, whether taken in the push notification or in the tool notifications menu.

When the user wants to acknowledge a notification, be it estimated time or check status, the workflow is very similar. The frontend sends a POST request to the endpoint `/notifications/ack` with the id of the notification. The notifications controller verifies that the user id that is encoded in the token attached to the request matches the user id associated with that notification. Afterward, it sends several queries to the database in order to delete the notification, edit the component associated with the notification and save the changes made in the history collection. Before sending the response to the frontend, the controller asks the database for the existing notifications for the user and returns them. The frontend updates the React `<Notifications Menu>` component. The sequence diagram described above is visible in Figure 5.25.

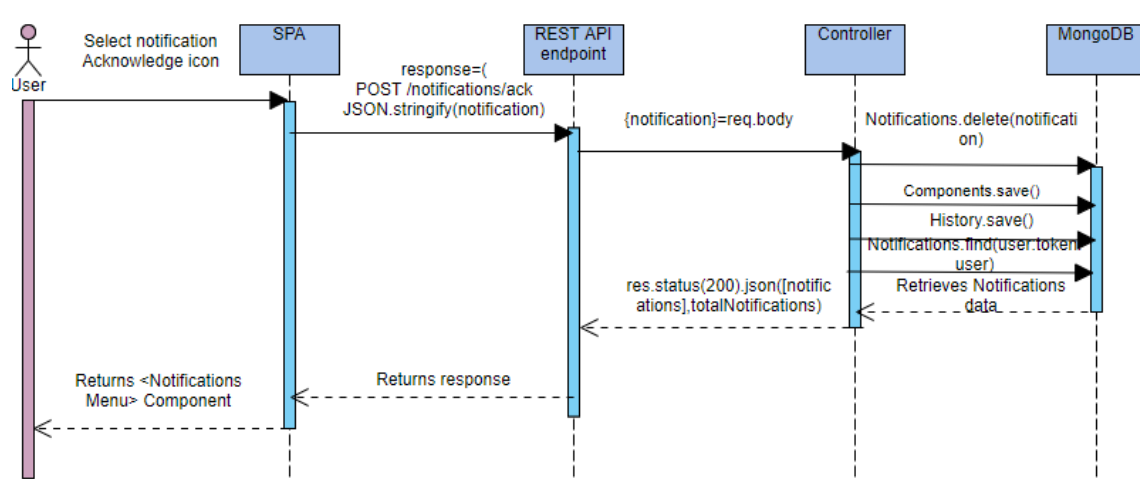


Figure 5.25: Acknowledge Notification Sequence Diagram

If the notification is a check status notification, the component's Ack attribute is updated to two months after the action date. If the notification is of the estimated time type, the component's Ack attribute is updated and the dueDate is extended for another two weeks. In both cases, a new entry is made in the History database table, describing that the component was edited.

The user, when wanting to acknowledge an overtime type notification, must first confirm the action in the dialog message that appears in the UI. Then, the frontend sends a POST request to the endpoint `/notifications/ackot` with the id of the notification. The notification controller sends several queries to the database in order to update the `dueDate` and `Ack` of the component associated with the notification, eliminate all notifications associated with that component, and add to the History table that the component was edited. In response, it sends the remaining notifications of that user to the frontend. The frontend renders the `<Notifications Menu>` component. The sequence diagram described above is visible in Figure 5.26.

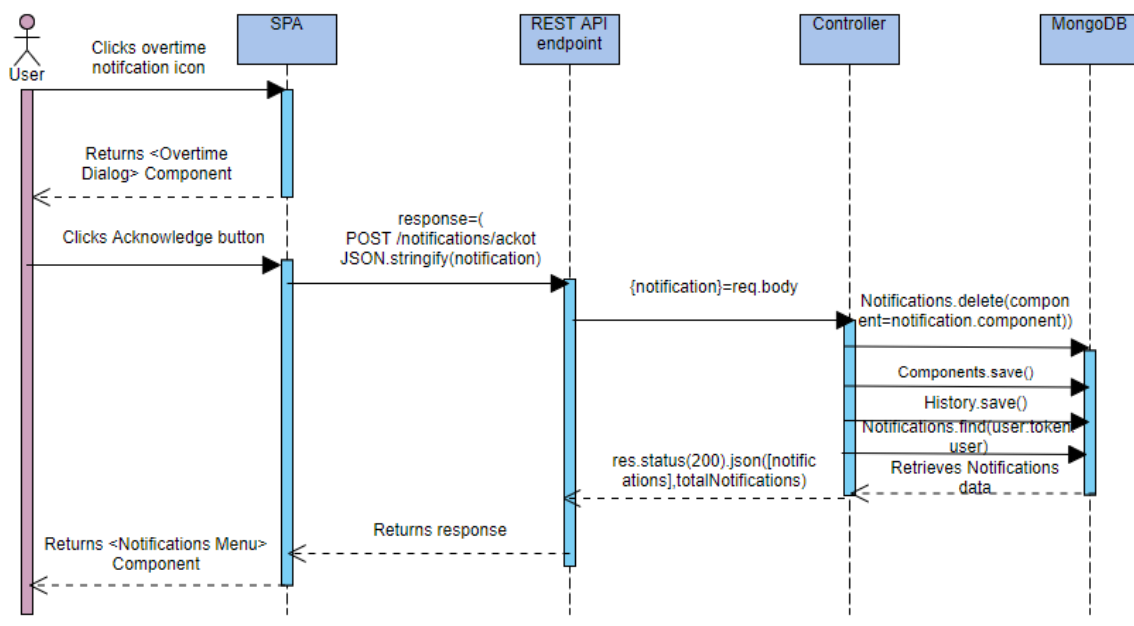


Figure 5.26: Acknowledge Over Time Notification Sequence Diagram

If the user chooses to clear the overtime notification instead of acknowledging it, the process is the same as for other notification types and is explained next.

When the user wants to clear a notification, the workflow is very similar in all notification types. The frontend sends a POST request to the endpoint `/notifications/clear` with the id of the notification. The notifications controller verifies that the user id that is encoded in the token attached to the request matches the user id associated with that notification. Afterward, it sends several queries to the database in order to delete the notification, edit the component associated with the notification and save the changes made in the history collection. Before sending the response to the frontend, the controller asks the database for the existing notifications for the user and returns them. The frontend updates the React `<Notifications Menu>` component. The sequence diagram described above is visible in Figure 5.27.

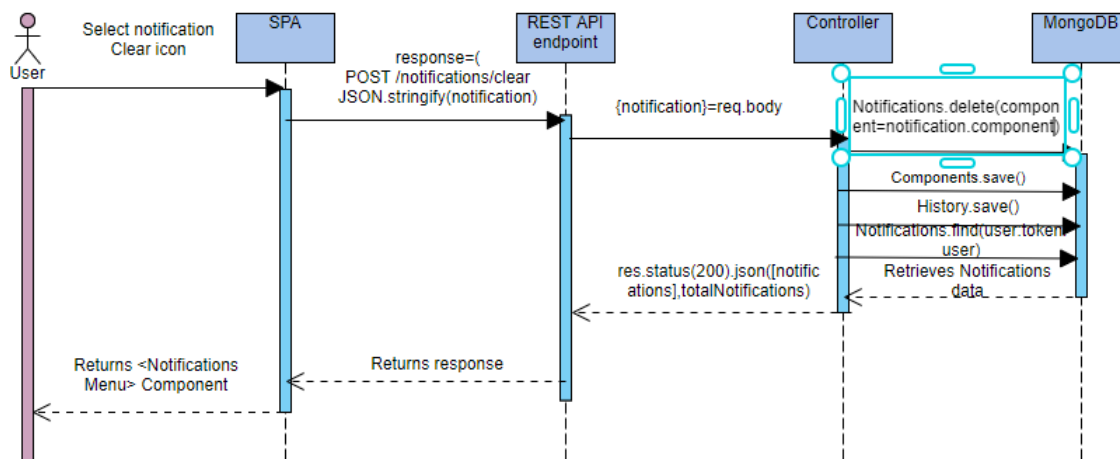


Figure 5.27: Clear Notification Sequence Diagram

In any type of notification, the component is edited and becomes free (status = available) and all notifications related to that component are deleted. In the History collection, it is added that the component was deleted and on what date.

When the user wants to acknowledge all check status and estimated time notifications simultaneously, he/she clicks on the respective icon that appears in the tool's notifications menu. Then, the frontend sends a POST request to the endpoint /notifications/ackall with the id of the notifications. The notifications controller verifies that the user id that is encoded in the token attached to the request matches the user id associated with the notifications. Afterward, it sends several queries to the database in order to delete the notifications, edit the components associated with the notifications and save the changes made in the history collection. Before sending the response to the frontend, the controller asks the database for the existing notifications for the user and returns them. The frontend updates the React <Notifications Menu> component. The sequence diagram described above is visible in Figure 5.28.

In summary, the system was successfully implemented. The system setup was defined and the third-party libraries integrated were described. Then, the developed features were presented.

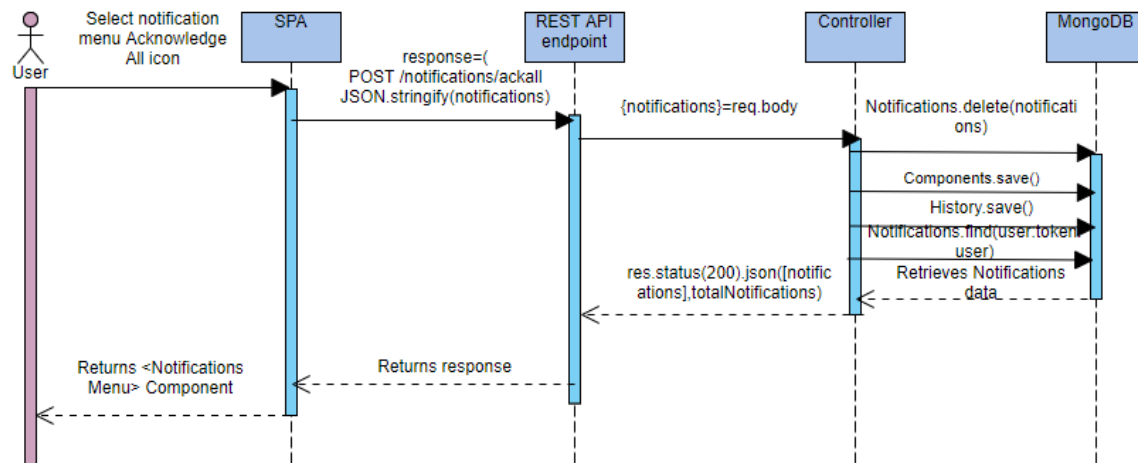


Figure 5.28: Acknowledge All Notifications Sequence Diagram

Chapter 6

Testing and Validation

In this chapter, the testing frameworks used throughout the project are presented and some example tests are also shown.

6.1 API Testing

Postman is a popular API client that makes it easy for developers to create, share, test, and document APIs. This is done by allowing users to generate and save simple and sophisticated HTTPS queries, as well as read their answers. As a result, work is more efficient and less tedious.

Postman was used to test the backend without necessarily having to send HTTP requests through the frontend. Thus, the backend could be tested and developed autonomously. Automated tests were also implemented to ensure that any changes to the backend code would not break the developed REST API.

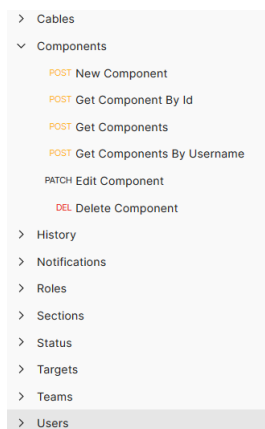


Figure 6.1: Postman Collections

In Figure 6.1 it is possible to see effectively all the defined API endpoints. Thus, it was possible to quickly and simply test all the endpoints of the REST API built for the project. In Figure 6.2 it is possible to see a GET request for the endpoint `/users/` and the corresponding response:

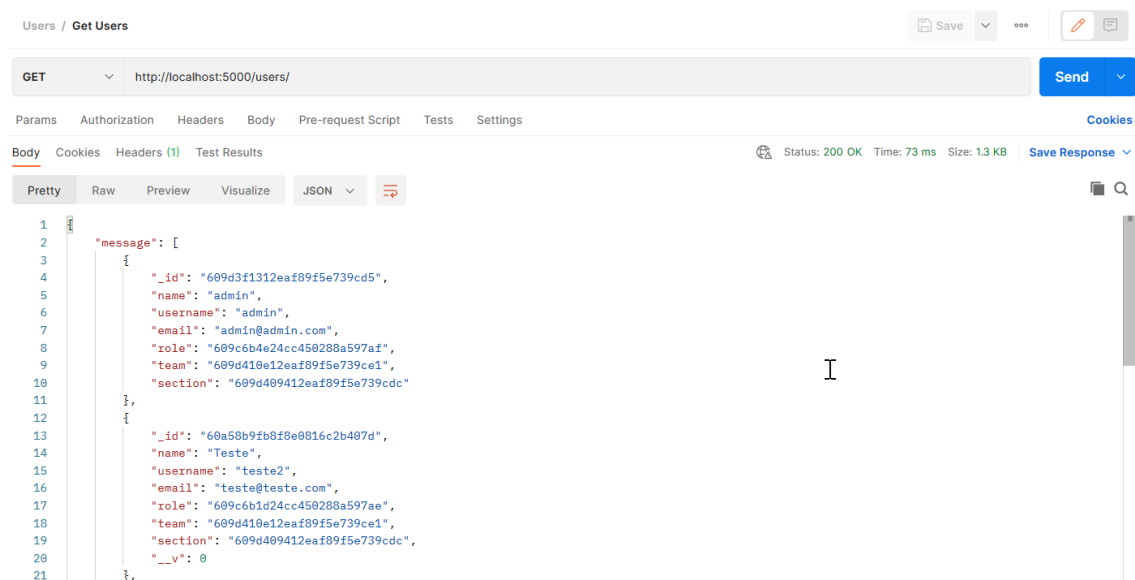


Figure 6.2: Postman Request

These tests were essential for the development of the project because besides being possible to validate the server-side architecture, it quickened the development of the project since it was not necessary to use the frontend to test the backend.

6.2 End to End Testing

End to End (E2E) testing usually visits the application in a browser and performs actions via the UI just like a real user would. The goals of E2E are to check the software from the user perspective, to avoid malfunctions caused by a failure of any of the subsystems, and if everything is working as expected and flowing smoothly.

Cypress is an all-in-one tool that allows performing E2E tests. It is possible to write tests, run tests, watch and record tests and also debug tests. Cypress was used in this project primarily because it is easy to write tests and it is possible to view live tests running. It was also possible to automatically generate an HTML report, as shown in Figure 6.3.

As with Postman, it was easy to ensure that the API was functional, Cypress was essential for validating the frontend code. Whenever new code was developed, E2E tests were performed to check if this new code had negatively impacted the expected results and quickly detect the issue before it became problematic.

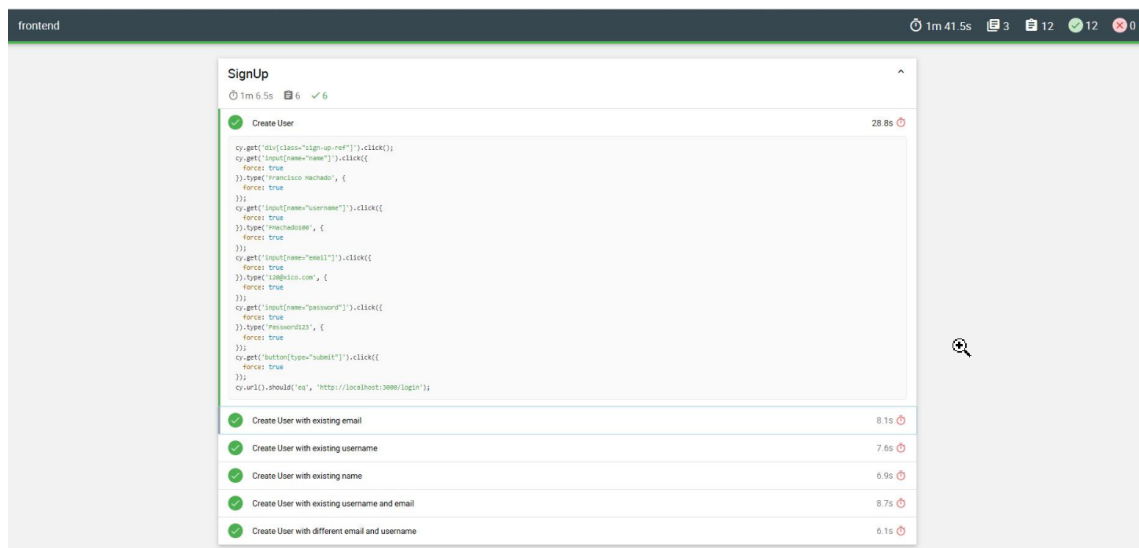


Figure 6.3: Cypress HTML report

In summary, the system validation was achieved thanks to the Postman and Cypress testing frameworks.

Chapter 7

Conclusion

In this last chapter, a summary of the work done is presented in section 7.1. Afterwards, in section 7.2, the initially proposed objectives are discussed. Finally, in section 7.3 further system improvements are proposed.

7.1 Summary

The work carried out on this project was quite exhaustive. It started with the analysis and discussion of the system requirements desired by the client. Afterward, market research regarding tools that had similar goals was done. As it was necessary to develop an application from scratch, the different technologies and architectures that exist for the full-stack development of a web application were considered. After choosing the development technologies it was needed to learn to work with them. After, concepts of agile methodology were learned to follow the company work methodology. Several UML diagrams, UI mockups were developed, and the API Endpoints were. Finally, the application was developed from scratch and autonomously and two testing tools were also used to ensure the integrity of the developed solution.

7.2 Objective Fulfillment

To evaluate the objective fulfillment two metrics are considered: the goals defined by the client and described in chapter 1.3, and the characteristics by which the tools were evaluated in chapter 2.1.

Table 7.1 was drawn with the goals defined by the client as well as their degree of achievement on a scale of 1 to 5, with level 1 meaning that it was not possible to achieve this objective at all and level 5 meaning that it was fully achieved.

In general, as shown in Table 7.1, the goals defined by the client were successfully achieved. The system's architecture helped achieve these goals easily: React and Material-UI provide a

Table 7.1: Goals and Achievement Degree

Goal	Achievement Degree
Have a user-friendly tool available to be used by the team	5
Have the inventory stored in a server and accessible to all team members	5
Manage the information and share it with the user when requested	5

friendly UI, MongoDB stores all the data, and a well-built REST API ensures that the user can request any information.

Table 7.2 was built in order to evaluate the tool developed, taking into account the features used to compare the existing solutions.

Table 7.2: Infrastructure Management Tool Features

Tool	User Authentication	User Management	Roles	Asset Information	Asset History	Notification System	Search Engine	Bosch Integration	Price
Infrastructure Management Tool	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Free

The tool developed implements all the evaluated features, as shown in Table 7.2. Furthermore, it was free and its scalability does not entail any additional costs for the company. This platform is also highly customizable by the enterprise.

In summary, the initially proposed objectives were met. The work carried out was relevant both for the company and for me. Throughout this project, I managed to learn new technologies related to web development and at the same time implement them in a real project, developed in a real work environment and using agile methodologies. The company got a much needed solution.

7.3 Future Work

Although essentially all goals reached a very satisfying level, there is margin for improvement. These system improvements are covered by the following topics:

- **Reservation System:** The reservation system is a nice to have requirement, but unfortunately there was not enough time for developing this functionality. This feature adds value to the developed tool because it allows the user to reserve a component that they want to use in the near future.
- **Performance and Security Testing:** In order to make the tool more robust, it is necessary to carry out performance and security tests. These tests typically assess responsiveness and speed under a workload and also allows to judge system vulnerabilities against different types of security attacks.

- **Bosch Integration:** The tool was developed as a proof of concept. The next step is to integrate into the Bosch services.
- **User Guide:** The user guide is intended to support the new users in how to use the Infrastructure Management Tool.

Appendix A

Appendix

A.1 Prices

	ESSENTIAL	ADVANCED	PREMIUM
ITEMS	250	250	250
PRICE (PER MONTH,BILLED ANUALLY)	\$35,00	\$45,00	\$50,00
USERS	500	500	500
FEATURES	• ASSET TRACKING • RESERVATIONS • ALERTS • MAINTENANCE • BASIC REPORTING	ALL ESSENTIAL FEATURES AND: • SCHEDULED SERVICES • ADVANCED REPORTING • CARTS • CUSTOM FIELDS • PURCHASE ORDERS • API INTEGRATION	ALL ADVANCED FEATURES AND: • ADVANCED ACESS CONTOL • CUSTOM SUBSTATES • SERVICE DESK INTEGRATIONS • WORK ORDERS AND PROJECT VIEW • SSO & G SUITE INTEGRATION

Figure A.1: EZOfficeInventory Plans and Prices with 250 items

A.2 UI Mockup

	ESSENTIAL	ADVANCED	PREMIUM
ITEMS	14750	14750	14750
PRICE (PER MONTH,BILLED ANUALLY)	\$462,50	\$757,50	\$905,00
USERS	29500	29500	29500
FEATURES	• ASSET TRACKING • RESERVATIONS • ALERTS • MAINTENANCE • BASIC REPORTING	ALL ESSENTIAL FEATURES AND: • SCHEDULED SERVICES • ADVANCED REPORTING • CARTS • CUSTOM FIELDS • PURCHASE ORDERS • API INTEGRATION	ALL ADVANCED FEATURES AND: • ADVANCED ACCESS CONTROL • CUSTOM SUBSTATES • SERVICE DESK INTEGRATIONS • WORK ORDERS AND PROJECT VIEW • SSO & G SUITE INTEGRATION

Figure A.2: EZOfficeInventory Plans and Prices with 14750 items

A.3 Extra Application UI

NUMBER OF IT ASSETS	PRICE (ANNUALLY)
250	\$955
500	\$1795
1000	\$2995
1500	\$4195
2000	\$4795
3000	\$7195
5000	\$9595
10000	\$11995

Figure A.3: ManageEngine AssetExplorer Subscription Model

	Self-Hosted	Monthly Hosted	Annual Hosted	Support-Only	Enterprise Support
Price	Free	\$39.99 USD/month	\$399.99 USD/year	\$449.99 USD/year	\$4,999.99 USD/year
Max Users	∞	∞	∞	∞	∞
Max Assets	∞	∞	∞	∞	∞
Github Support	✓	✓	✓	✓	✓
Email Support	✗	✓	✓	✓	✓
Priority Feature Requests	✗	✓	✓	✓	✓
API Integration Help	✗	✗	✗	✗	✓
Phone Support	✗	✗	✗	✗	✓
TLS (SSL) Included	✗	✓	✓	--	--
Automatic Backups	✗	✓	✓	--	--
Automatic Upgrades	✗	✓	✓	--	--
Server Maintenance	✗	✓	✓	--	--

Figure A.4: Snipe-IT Pricing
[10]

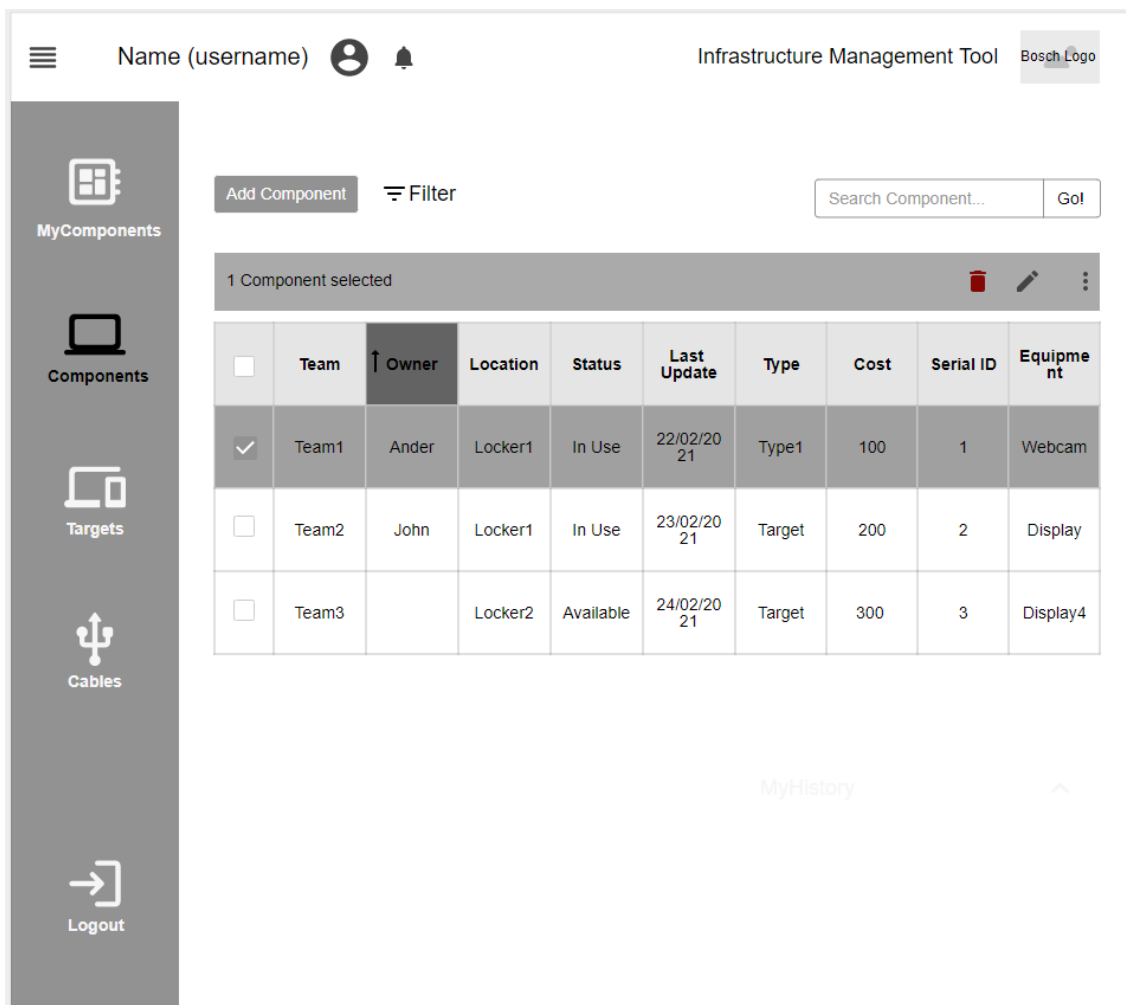


Figure A.5: Components Catalog UI Mockup

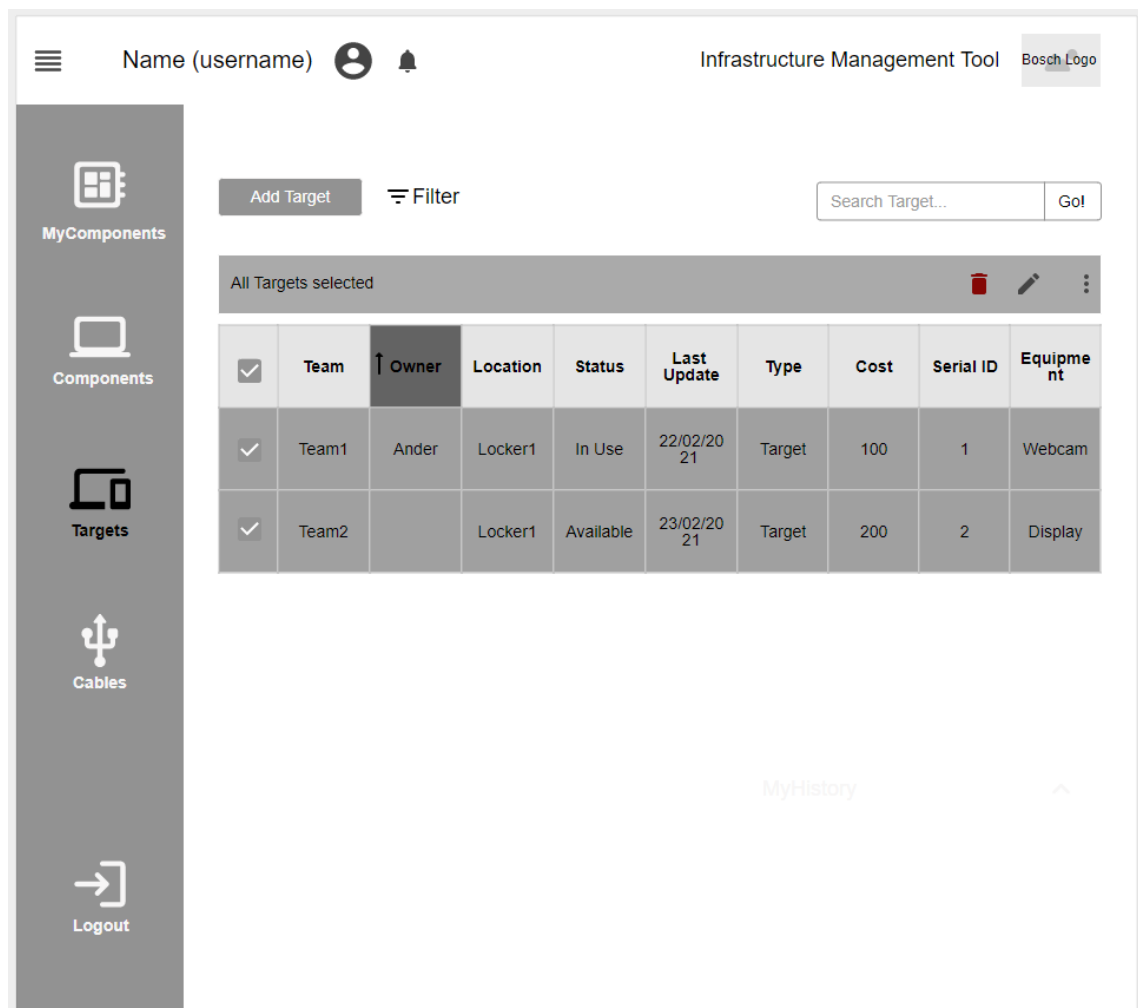


Figure A.6: Targets Catalog UI Mockup

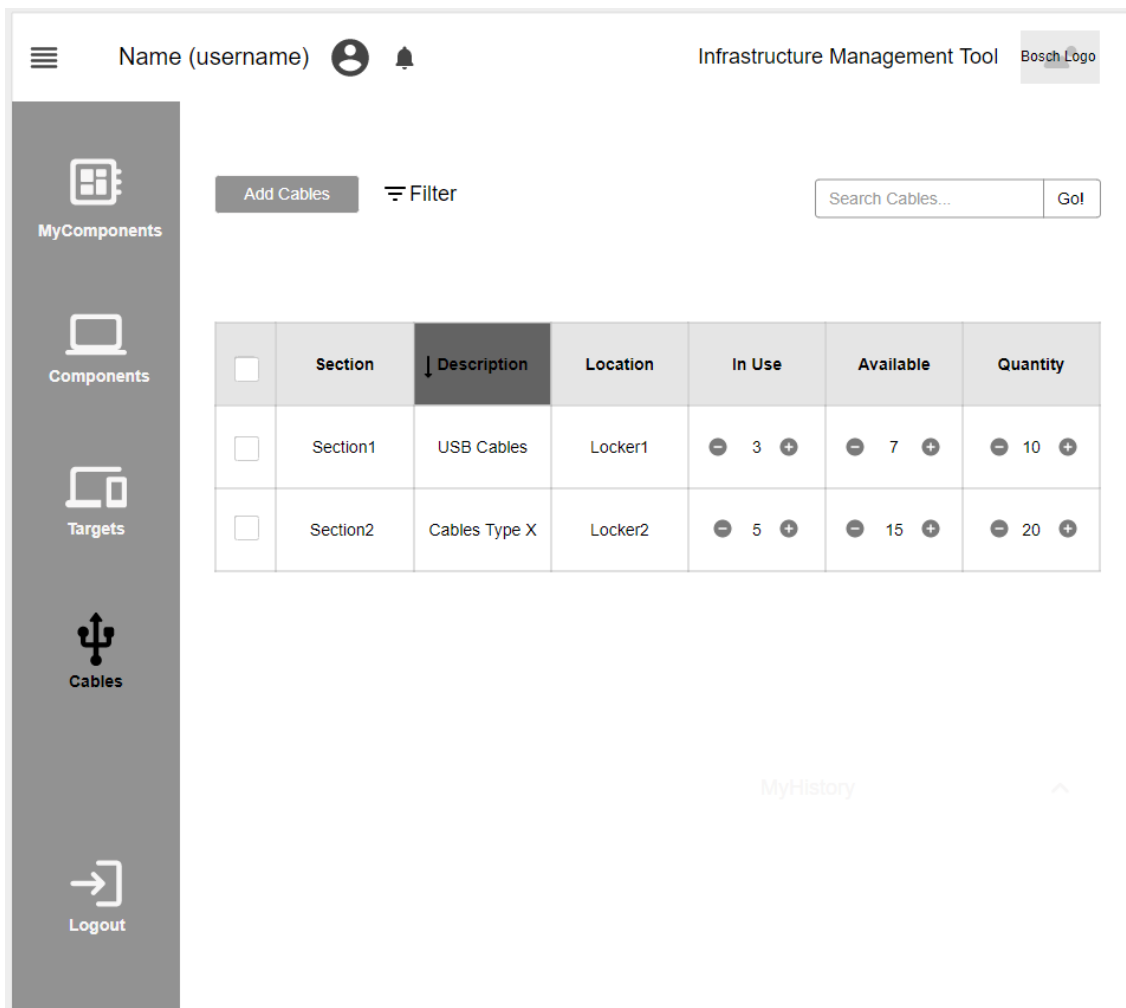


Figure A.7: Cables Catalog UI Mockup

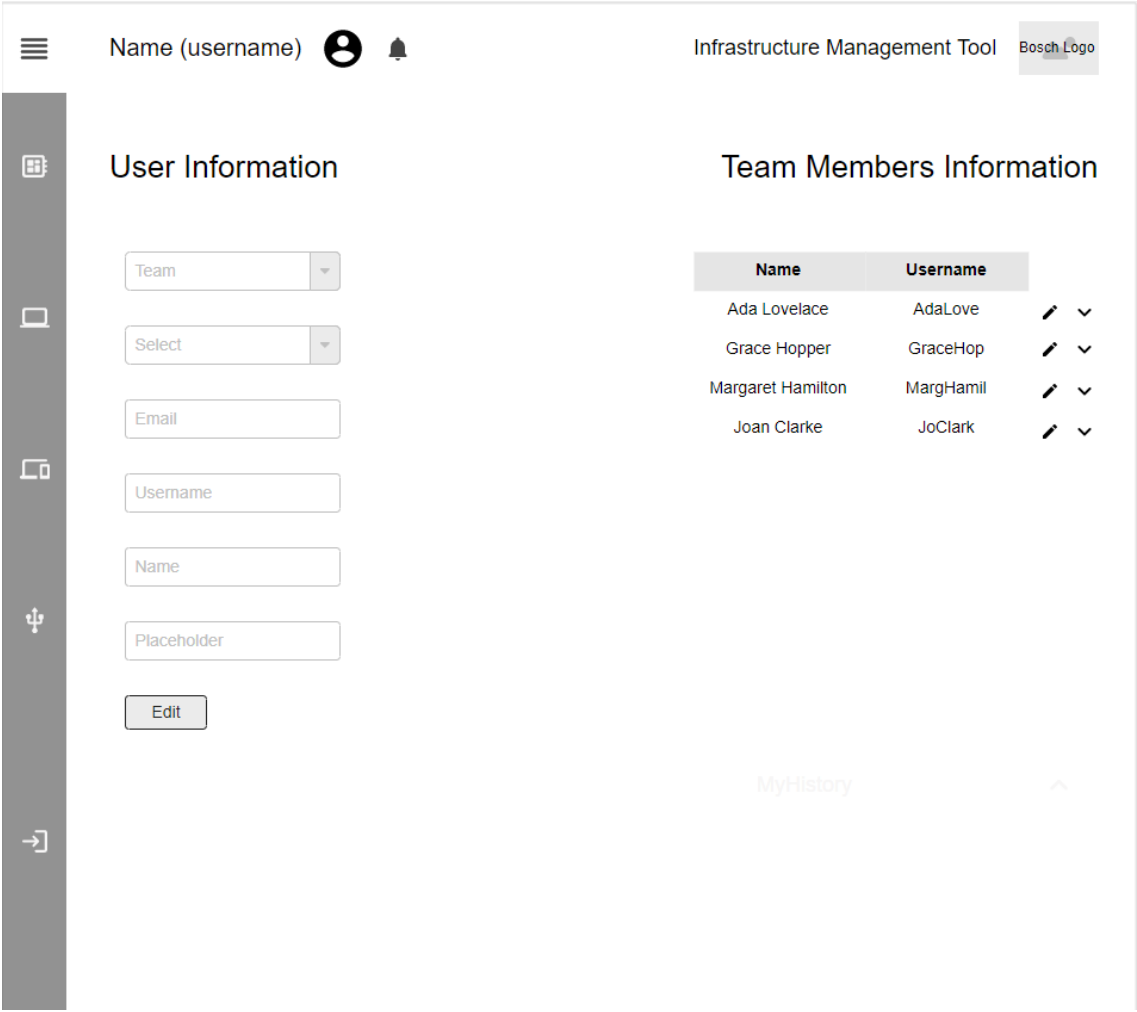


Figure A.8: User Information UI Mockup

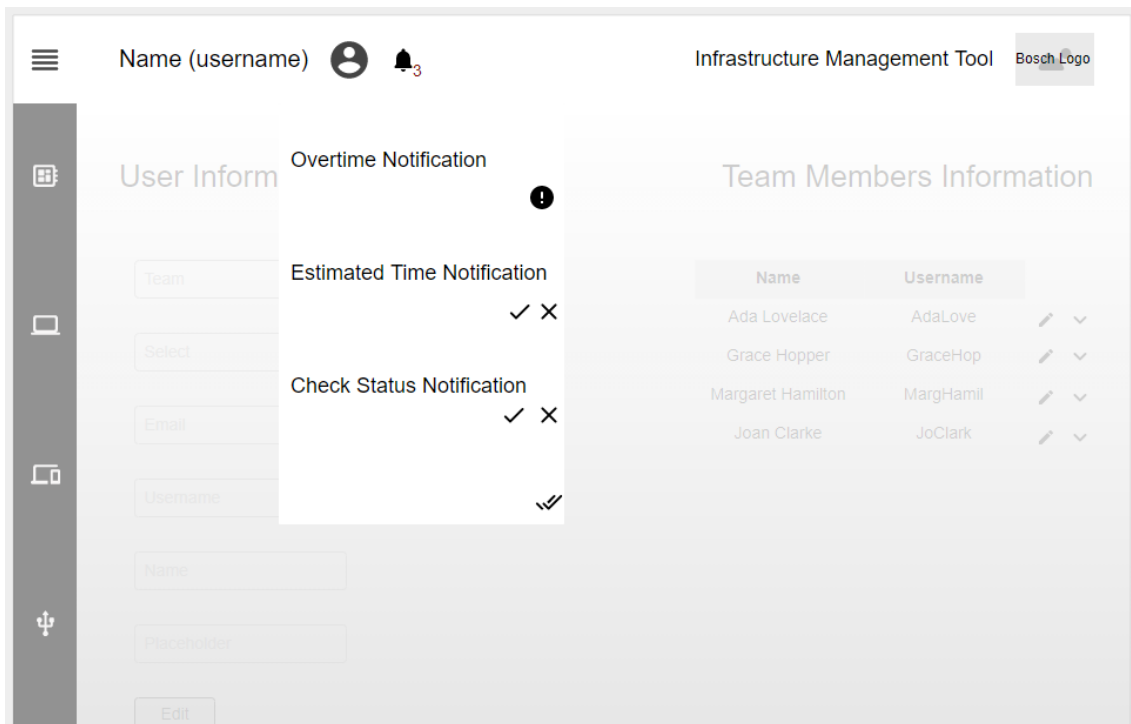


Figure A.9: Notifications Menu UI Mockup

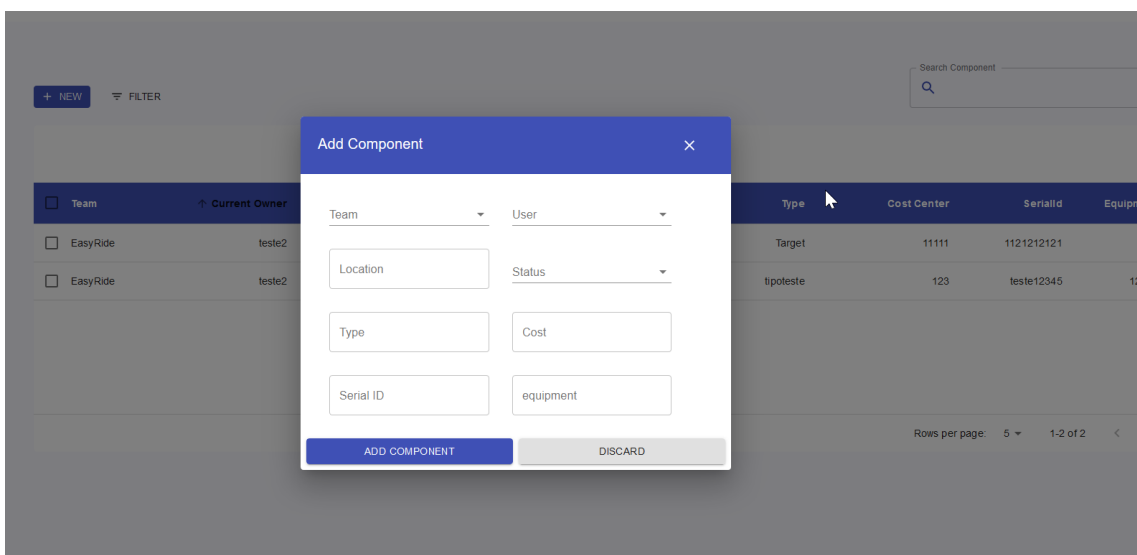


Figure A.10: Add Component UI

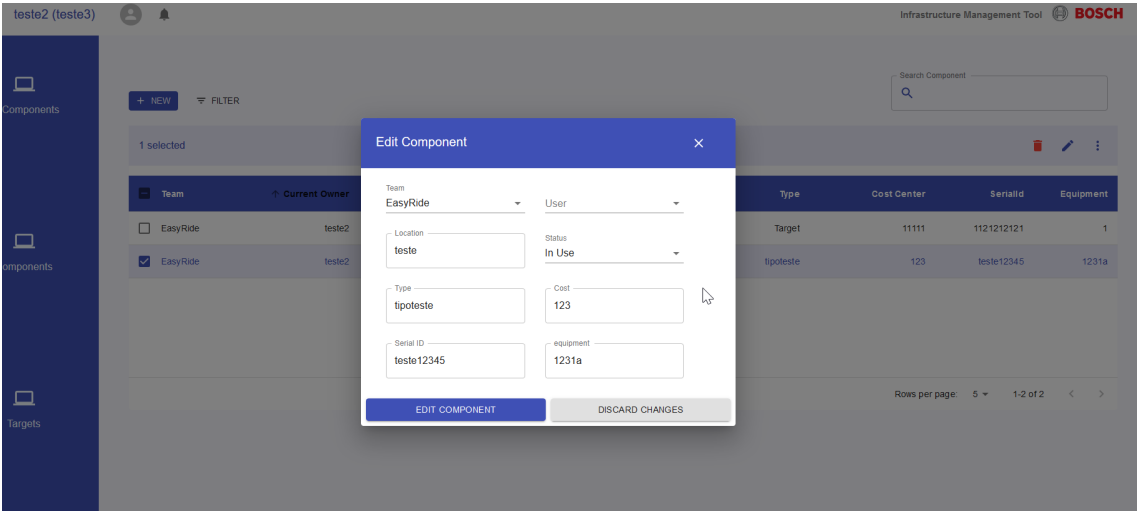


Figure A.11: Edit Component UI

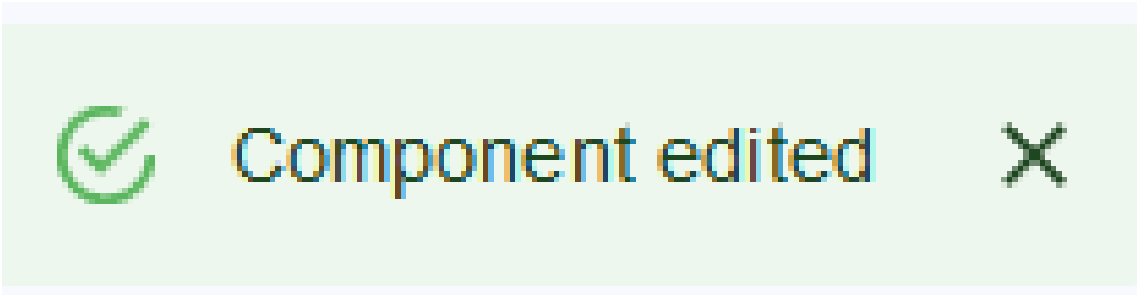


Figure A.12: Success Message UI



Figure A.13: Delete Message UI

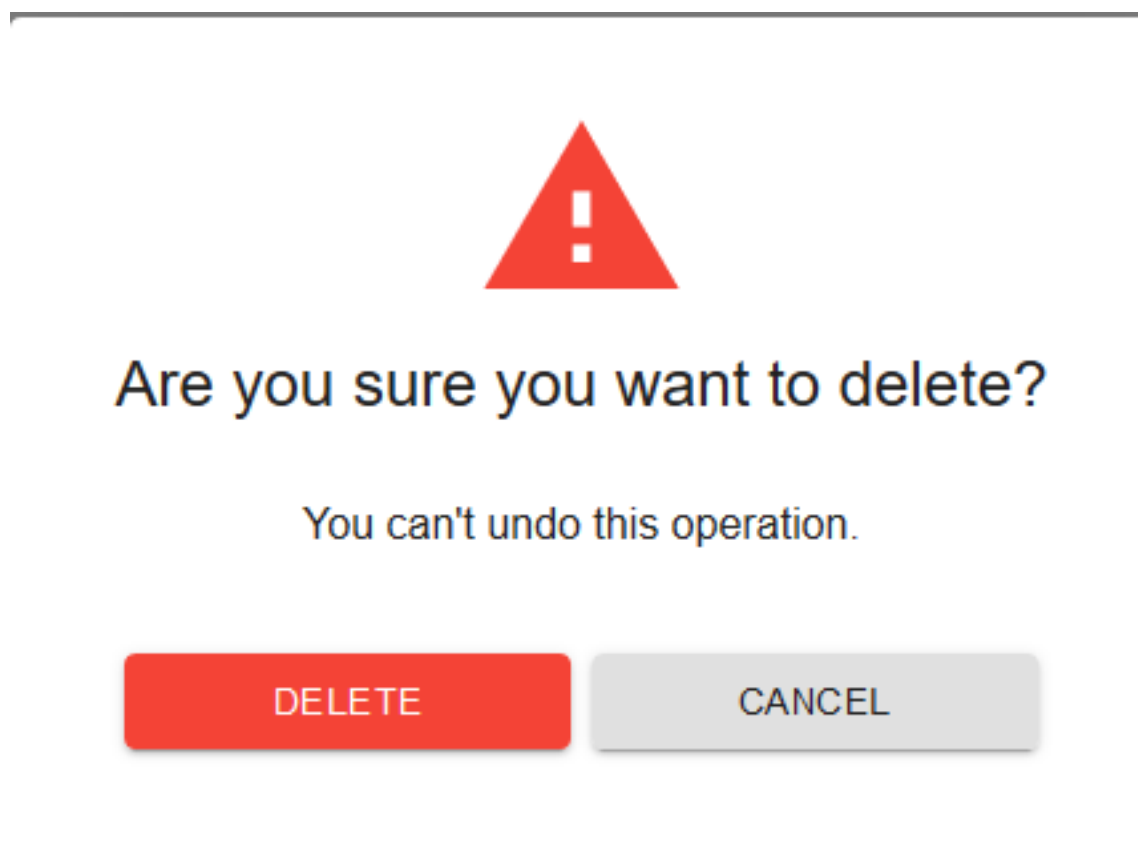


Figure A.14: Confirmation Dialog UI

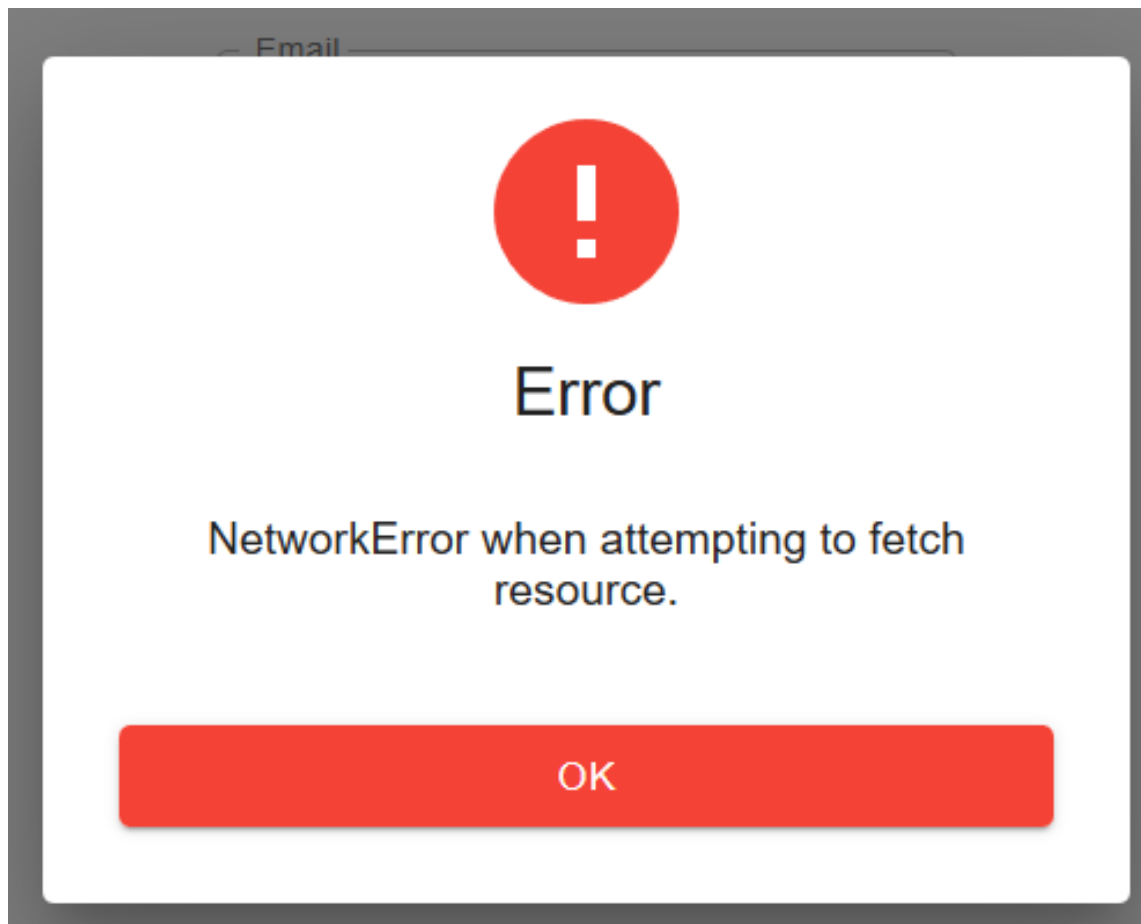


Figure A.15: Error Modal UI

References

- [1] Bosch. Bosch - About Us. URL: <https://www.bosch.com/company/>.
- [2] Bosch. Bosch em Portugal. URL: <https://www.bosch.pt/a-nossa-empresa/bosch-em-portugal/braga/>.
- [3] Chip Galusha. Getting Started with IT Asset Management. Technical report, 2001.
- [4] Service Now. IT Asset Management, 2020. URL: <https://www.servicenow.com/content/dam/servicenow-assets/public/en-us/doc-type/resource-center/data-sheet/ds-hardware-asset-management-orlando.pdf>.
- [5] Service Now. Asset Management. URL: <https://www.servicenow.com/products/asset-management.html>.
- [6] EZ Office Inventory. EZ Office Inventory, 2021. URL: <https://www.ezofficeinventory.com/>.
- [7] ManageEngine. ManageEngine AssetExplorer, 2021. URL: <https://www.manageengine.com/products/asset-explorer/>.
- [8] Lansweeper. Lansweeper, 2021. URL: <https://www.lansweeper.com/>.
- [9] Asset Panda. Asset Panda, 2021. URL: <https://www.assetpanda.com/>.
- [10] Snipe-IT. Snipe-IT, 2021. URL: <https://snipeitapp.com/>.
- [11] Spiceworks. Spiceworks, 2021. URL: https://www.spiceworks.com/free-pc-network-inventory-software/?utm_medium=internal&utm_source=community_tools&utm_campaign=community_tools03/06#overview.
- [12] TechTerms. Frontend, 2020. URL: <https://techterms.com/definition/frontend>.
- [13] Palaksinghal9903. Frontend vs Backend, 2021. URL: <https://www.geeksforgeeks.org/frontend-vs-backend/>.
- [14] Patrícia Pereira. Single-page applications, 2019. URL: <https://www.xpand-it.com/blog/single-page-applications/>.
- [15] Aaron Groom. What Is a Single-Page Application?, 2018. URL: <https://dzone.com/articles/what-is-a-single-page-application>.

- [16] Lvivity. Single-page App vs. Multi-page App: Pros, Cons, and Which is Better?, 2021. URL: <https://lvivity.com/single-page-app-vs-multi-page-app>.
- [17] Arvind Padmanabhan. Single Page Application, 2020. URL: <https://devopedia.org/single-page-application>.
- [18] HUSPI. What is a Single Page Application? Accessible Guide to SPA, 2019. URL: <https://huspi.com/blog-open/definitive-guide-to-spa-why-do-we-need-single-page-applications/>.
- [19] Adrej Gajdos. Single-Page Application vs Multiple-Page Application: Which One To Choose For Your Project, 2020. URL: <https://andrejgajdos.com/single-page-application-vs-multiple-page-application/>.
- [20] Deepak Masheshwari. Why Single Page Application (SPA) architecture is so popular?, 2021. URL: <https://medium.com/nerd-for-tech/why-single-page-application-spa-architecture-is-so-popular-141b85400204>.
- [21] Maria Tsarouva. The pros and cons of single page applications (SPAs), 2019. URL: <https://www.itechart.com/blog/pros-cons-of-single-page-applications/>.
- [22] Angular. What is Angular?, 2021. URL: <https://angular.io/guide/what-is-angular>.
- [23] Angular. Introduction to Angular concepts, 2021. URL: <https://angular.io/guide/architecture>.
- [24] Bhavika Garg. Angular Architecture Overview, 2019. URL: <https://medium.com/@bhavikagarg8/angular-architecture-overview-1e7cc7483a0>.
- [25] Chinmayee Deshpande. What is Angular?: Architecture, Features, and Advantages. URL: <https://www.simplilearn.com/tutorials/angular-tutorial/what-is-angular>.
- [26] Besant Technologies. What is Angular?, 2021. URL: <https://www.besanttechnologies.com/what-is-angular>.
- [27] Ari Lerner. What is React?, 2018. URL: <https://www.newline.co/fullstack-react/30-days-of-react/day-1/>.
- [28] Jamsheer K. Features of React JS: Main Pros and Cons, 2019. URL: https://acowebs.com/react-js-features/#Features_of_ReactJS.
- [29] Taha Sufiyan. What is ReactJS: Introduction To React and Its Features, 2021. URL: <https://www.simplilearn.com/tutorials/reactjs-tutorial/what-is-reactjs>.
- [30] Scott Morris. Tech 101: What Is React JS?, 2021. URL: <https://skillcrush.com/blog/what-is-react-js/#what>.
- [31] ThemeSelection. Vue Ecosystem, 2021. URL: <https://medium.com/js-dojo/vue-ecosystem-979773a9bf54>.
- [32] Vue.js. What is Vue.js?, 2021. URL: <https://vuejs.org/v2/guide/>.

- [33] Npmtrends. NPM trends, 2021. URL: <https://www.npmtrends.com/react-vs-vue-vs-@angular/core>.
- [34] Stackoverflow. Stack Overflow Trends, 2021. URL: <https://insights.stackoverflow.com/trends?tags=reactjs%2Cvue.js%2Cangular%2Cangularjs>.
- [35] Angular. Angular GitHub, 2021. URL: https://github.com/angular/angular/network/dependents?package_id=UGFja2FnZS00NTE2NDYyMzQ%3D.
- [36] Facebook. React GitHub, 2021. URL: https://github.com/facebook/react/network/dependents?package_id=UGFja2FnZS0xMzM2NDkxNg%3D%3D.
- [37] Vuejs. Vue GitHub, 2021. URL: https://github.com/vuejs/vue/network/dependents?package_id=UGFja2FnZS00OTM3Mjg3MDY%3D.
- [38] Aris Pattakos. Angular vs React vs Vue 2021, 2021. URL: <https://athemes.com/guides/angular-vs-react-vs-vue/>.
- [39] React. Testing Overview, 2021. URL: <https://reactjs.org/docs/testing.html>.
- [40] Angular. Testing, 2021. URL: <https://angular.io/guide/testing#set-up-testing>.
- [41] Nelly Nelson. Angular vs. React vs. Vue, 2020. URL: <https://medium.com/swlh/angular-vs-react-vs-vue-802a7c5f7e50>.
- [42] Krausest. Performance, 2021. URL: <https://krausest.github.io/js-framework-benchmark/current.html>.
- [43] Shaumik Daityari. Angular vs React vs Vue: Which Framework to Choose in 2021, 2021. URL: <https://www.codeinwp.com/blog/angular-vs-vue-vs-react/>.
- [44] Guru99. Typescript vs JavaScript: What's the Difference?, 2021. URL: <https://www.guru99.com/typescript-vs-javascript.html#9>.
- [45] Mariana Berga and Rute Figueiredo. TYPESCRIPT VS JAVASCRIPT: WHICH ONE IS BETTER?, 2021. URL: <https://www.imaginarycloud.com/blog/typescript-vs-javascrrip/#js>.
- [46] Bishal Kumar Dubey. Difference between TypeScript and JavaScript, 2020. URL: <https://www.geeksforgeeks.org/difference-between-typescript-and-javascript/>.
- [47] Codecademy. Back-End Web Architecture. URL: <https://www.codecademy.com/articles/back-end-architecture>.
- [48] Facundo Giuliani. GraphQL vs REST: which API is best for your web app?, 2021. URL: <https://www.sanity.io/guides/graphql-vs-rest-api-comparison#ce0f4f315367>.
- [49] MuleSoft. What is a REST API? URL: <https://www.mulesoft.com/resources/api/what-is-rest-api-design>.

- [50] Red Hat. What is a REST API?, 2020. URL: <https://www.redhat.com/en/topics/api/what-is-a-rest-api>.
- [51] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, UNIVERSITY OF CALIFORNIA, IRVINE, 2020. URL: [https://www.ics.uci.edu/~sim\\$fielding/pubs/dissertation/rest_arch_style.htm](https://www.ics.uci.edu/~sim$fielding/pubs/dissertation/rest_arch_style.htm).
- [52] Alexander S. Gillis. REST API (RESTful API), 2020. URL: <https://searchapparchitecture.techtarget.com/definition/RESTful-API>.
- [53] João Inez. GRAPHQL VS REST: WHICH ONE IS BETTER? URL: <https://www.imaginarycloud.com/blog/graphql-vs-rest/#Comparison>.
- [54] GraphQL. GraphQL, 2018. URL: <https://spec.graphql.org/June2018/#sec-Overview>.
- [55] GraphQL. GraphQL, 2021. URL: <https://graphql.org/>.
- [56] Altexsoft. GraphQL: Core Features, Architecture, Pros and Cons, 2019. URL: <https://www.altexsoft.com/blog/engineering/graphql-core-features-architecture-pros-and-cons/>.
- [57] Duomly. REST API vs. GraphQL [comparison], 2020. URL: <https://dev.to/duomly/rest-api-vs-graphql-comparison-3j6g>.
- [58] Chimezie Enyinnaya. REST versus GraphQL, 2017. URL: <https://blog.pusher.com/rest-versus-graphql/>.
- [59] David Feuer. GraphQL: Building a consistent approach for the API consumer, 2021. URL: <https://cloud.google.com/blog/products/api-management/interacting-with-apis-rest-and-graphql>.
- [60] Devopedia. GraphQL, 2019. URL: <https://devopedia.org/graphql>.
- [61] Mary B. Prescott Fred R. Mcfadden, Jeffrey A. Hoffer. *Modern Database Management*. 2003.
- [62] Oracle. What Is a Database?, 2021. URL: <https://www.oracle.com/database/what-is-database/>.
- [63] Samuel Sam. Types of databases, 2018. URL: <https://www.tutorialspoint.com/Types-of-databases>.
- [64] Prentice Hall. *Management Information Systems*. 2006. URL: [https://paginas.fe.up.pt/~sim\\$acbritto/laudon/ch7/chpt7-2main.htm](https://paginas.fe.up.pt/~sim$acbritto/laudon/ch7/chpt7-2main.htm).
- [65] MongoDB. MongoDB Architecture Guide: Overview. 2020. URL: https://info-mongodb-com.s3.us-east-1.amazonaws.com/MongoDB_Architecture_Guide.pdf.
- [66] MongoDB. Bosch Leads Charge into Internet of Things, 2021. URL: <https://www.mongodb.com/customers/bosch>.

- [67] Kate Eby. Comprehensive Guide to the Agile Manifesto, 2016. URL: <https://www.smartsheet.com/comprehensive-guide-values-principles-agile-manifesto>.
- [68] Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, and Dave Thomas. Manifesto for Agile Software Development, 2001. URL: <https://agilemanifesto.org/>.
- [69] Scrum.org. WHAT IS SCRUM?, 2021. URL: <https://www.scrum.org/resources/what-is-scrum>.
- [70] Ken Schwaber and Jeff Sutherland. *The Scrum Guide*. 2020. URL: <https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf#zoom=100>.
- [71] Kanbanize. What is Kanban?, 2021. URL: <https://kanbanize.com/kanban-resources/getting-started/what-is-kanban>.
- [72] strongback. rtc-user-story-example, 2016. URL: <https://www.strongback.us/wp-content/uploads/2016/03/rtc-user-story-example-2-1.jpg>.
- [73] jazz. scrum-taskboard-full. URL: <https://jazz.net/agile/images/scrum-taskboard-full.png>.
- [74] Manuel Lorenz and Maximilian Schwarzmuller. React, NodeJS, Express MongoDB - The MERN Fullstack Guide, 2020. URL: <https://www.udemy.com/course/react-nodejs-express-mongodb-the-mern-fullstack-guide/>.
- [75] Expressjs. Express, 2021. URL: <https://expressjs.com/>.
- [76] Auth0. Introduction to JSON Web Tokens, 2021. URL: <https://jwt.io/introduction>.