FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Web Application for documenting and tracking of hardware and software operations on production equipment

Nuno Minhoto

FINAL VERSION

Mestrado em Engenharia Eletrotécnica e de Computadores

Supervisor: Professor Armando Sousa

August, 2022

This work was carried out with the support of the host institution: Preh Portugal and supervised at the institution by: Hugo Oliveira and Paulo Oliveira.

Abstract

The purpose of the project associated with this dissertation is to develop a web application for Preh Portugal, as this is a dissertation in a business environment.

This company is dedicated to producing electrical components for the automotive industry. The industrialization department of the company is responsible for the creation of new devices for producing and testing those components. Those devices, including their hardware and software, need to follow and comply with the rules defined on a set of standards defined by the company when they are deployed for use.

The current method that is used by the company to store standards is through a Microsoft PowerPoint file, where every topic of a standard is documented, and for the users to verify if a certain equipment complies with the rules stated on the standard, they have to go thought the whole document searching for the correct ones, create an Microsoft Excel spreadsheet with all the needed rules, and then use that document as a check list to verify and approve if all rules are up to standard. This whole process can be cumbersome, and adds an extra unwanted effort when approving equipment.

This dissertation describes the process of planing and developing the creation of a new platform to improve on the method described above method. The new platform has the goal to make the standards management process easier throughout a product design and production, as well as creating a new method that provides a pleasant user experience when consulting the rules of a standard, and when in need of approving equipment.

In order to achieve this, a requirements gathering was performed, and once it was done, the needed technologies where chosen considering the options available. Following this, the models and all the preparation work needed was performed before the development process started. The tools chosen to build the application was the Blazor framework joined with a Microsoft SQL server to use as a database. During the development stage all the features needed to comply with the requirements were built. After having the web application functional and deployed into the company server, it was allowed for the workers of the company to experiment and start using it. The feedback from those users was taken into consideration for adjustments and improvements to perform on future work for the web application. From that feedback, the general opinion of the end users was that they will most likely use this new platform instead of the old method.

The final result of the project associated with this dissertation was a web application that facilitates the process of storing the standards information and the approval operation, that is deployed into the company servers and can be accessed by some users. It is not on production stage yet but it will be put into production in a very near future.

Resumo

O objectivo do projecto associado a esta dissertação é desenvolver uma aplicação web para Preh Portugal, uma vez que se trata de uma dissertação num ambiente empresarial.

Esta empresa dedica-se à produção de componentes eletrónicos para a indústria automóvel. O departamento de industrialização da empresa é responsável pela criação de novos dispositivos para a produção e teste desses componentes. Esses dispositivos, incluindo o seu hardware e software, precisam de seguir e cumprir as regras definidas num conjunto de normas definidas pela empresa quando são implantados para utilização.

O método actual que é utilizado pela empresa para armazenar normas é através de um ficheiro Microsoft PowerPoint, onde cada tópico de uma norma é documentado, e para que os utilizadores verifiquem se um determinado equipamento cumpre as regras definidas na norma, têm de procurar em todo o documento procurando as correctas, criar uma folha de cálculo Microsoft Excel com todas as regras necessárias, e depois utilizar esse documento como uma lista de verificação para aprovar se todas as regras estão de acordo com as normas. Todo este processo pode ser incómodo, e acrescenta um esforço extra indesejado ao aprovar o equipamento.

Esta dissertação descreve o processo de planeamento e desenvolvimento da criação de uma nova plataforma para melhorar o método descrito acima. A nova plataforma tem o objectivo de facilitar o processo de gestão de normas ao longo da concepção e produção de um produto, bem como de criar um novo método que proporcione uma experiência agradável ao utilizador quando consultar as regras de uma norma, e quando necessitar de aprovação de equipamento.

Para o conseguir, foi realizada uma recolha de requisitos, e uma vez efectuada, as tecnologias necessárias foram escolhidas tendo em conta as opções disponíveis. Em seguida, os modelos e todo o trabalho de preparação necessário foram realizados antes de se iniciar o processo de desenvolvimento. As ferramentas escolhidas para construir a aplicação foram a framework Blazor unida a um servidor Microsoft SQL para ser utilizada como base de dados. Durante a fase de desenvolvimento, foram construídas todas as funcionalidades necessárias para cumprir os requisitos. Depois de ter a aplicação web funcional e implantada no servidor da empresa, foi permitido aos trabalhadores da empresa experimentar e começar a utilizá-la. O feedback desses utilizadores foi tomado em consideração para ajustes e melhorias a realizar no trabalho futuro para a aplicação web. A partir desse feedback, a opinião geral dos utilizadores finais foi que muito provavelmente irão utilizar esta nova plataforma em vez do método antigo.

O resultado final do projecto associado a esta dissertação foi uma aplicação web que facilita o processo de armazenamento da informação sobre as normas e a operação de aprovação, que está implantada nos servidores da empresa e que pode ser acedida por alguns utilizadores. Ainda não está em fase de produção, mas será colocado em produção num futuro muito próximo.

Acknowledgements

To Professor Armando Jorge Sousa, I appreciate and am thankful for the guidance and support throughout the development of this dissertation.

To the company Preh Portugal, everybody on the industrialization department and of course the software development team i would like to thank dearly for the opportunities that were given to me and for reception and all the good times during the development of the dissertation. I also have to appreciate the opportunities and support provided by Mr. Hugo Oliveira, leader of the software team, for hosting and being the responsible for the developed project. I would also like to thank Mr. Paulo Oliveira for everything he has done for me as a mentor, accompanying and helping me during all my time at the company.

Through the years I spent on FEUP there where good and bad moments, but they where all surpassed with the help of all the friends I met through the years, but most importantly with those that stayed with me from the beginning and all the ones that are from my hometown of Bragança. A big thank you to you all.

The last year would not be the same without the company of Francisco Parente and Beatriz Pinto. So thank you, for all the adventures, all the availability, support and everything else.

Lastly, i have to thank my family. Every last one of them, but most importantly, my father for always supporting me through my life, my brother that was always with me since i was born, and to my mother, since without her help and advice i would not have managed to get here.

Nuno Minhoto

"All we have to decide is what to do with the time that is given us."

J.R.R. Tolkien – The Fellowship of The Ring

Contents

List of Figures

Abbreviations and Symbols

API	Application programming interface
EF	Entity Framework
HMI	Human-machine interfaces
HTML	HyperText Markup Language
CSS	Cascading Style Sheets
JS	JavaScript
SPA	Single page application
UI	User interface
UX	User Experience
WASM	WebAssembly
JSON	JavaScript Object Notation
JWT	JSON web token
DB	Database
SQL	Structured Query Language
VPN	Virtual Private Network
MVC	Model-View-Controller

Chapter 1

Introduction

1.1 Context

The company to whom this product is being developed for, creates production equipment for the automotive industry. Every equipment produced by this enterprise has to follow certain rules defined in various standards created by them, in other to assure that each one of them will function according to expectations.

All the existing standards are defined in several files, one for each, with more than 80 pages and still growing. When a employee needs to do verification operations on equipment, he has to perform a lot of preparation work for the current used approval method.

This method, even tho it has been used by the company for quite some time, it is not a very efficient way to perform this task and it can be very tiring for the workers.

1.2 Idea and Motivations

The main motivation for this project can be found when the need to approve equipment arises. As explained above, is not easy to follow a document with 80 pages when approving equipment. It gets even harder if multiple documents are needed. The person that has to approve the equipment will have to search in various documents for the rules to apply.

In order to make this process faster and more efficient, as well as giving the employees a better experience while performing this task, the idea for a new application rose, and with it additional features to complement the whole process around approving equipment for a project.

This idea consists on a system that stores all the standards and their respective topics and rules. The system would be accessible by to the necessary workers, allowing them to search for the information stored regarding the standards. The users would also have the option to create new projects composed by several devices and equipment, which can be related to the standard rules previously stored. This way, when the time comes for the approval of equipment in a project, only the required rules corresponding to the ones present in that project will be shown from the standard and then the employee will only have to approve or deny that the device is built according to that

rule. After that, the system will keep a record of all the approval operations that were performed, and which user did them. This is an important feature since it allows for a history record to be kept, and if any problem occurs, all the information about the operation that might have caused it is recorded on the database.

1.2.1 Product Vision

- **Vision:** Making the process to approve equipment and the management and the search and management of standards easier.
- **Target group:** Preh Portugal industrialization department workers.
- **Needs:** Improving the user experience, and improving on the method that is currently being used.
- **Product attributes:** Simple to use, easy access, customizable, scalable, functional.
- **Time-frame:** July 2022

1.3 Requirements

The requirements listed in this chapter were identified by interviews with the supervising team and future potential users.

One of the requirements was to use the scrum framework for agile development.

Scrum is a framework for project management that can be applied to software development. It is designed for teams to break their work into goals that can be completed within fix timed iterations called sprints. At the end of each sprint, the team holds a meeting to demonstrate the work done and receive feedback called sprint review, and a sprint retrospective to reflect and improve on the previous sprint.

In order for the application of this framework to go smoothly and functional, there are three important roles that need to be assigned.

Product owner: Creates the project's overall requirements, objectives and release plan.

Scrum master: Is responsible for ensuring that Scrum values, practices, and rules are enacted and enforced, makes the link between management and the team and tries to remove any impediments imposed on developers.

Developers team: Is responsible for implementing the functionalities, the success or the failure of systems. Should be self-managing, self organizing, cross-functional and be committed to the project.

The actors involved in the project associated with this dissertation are:

- Developer - Nuno Minhoto

- Scrum master + developer - Paulo Oliveira
- Product owner - Hugo Oliveira
- Users - Potentially all engineers of the industrialization department of Preh

1.3.1 System needs

The following system needs were ascertained through conversations with the company responsible:

1. Centralize the texts of the relevant standards and rules on a single place (a database).
2. Easy access to the standards and rules information.
3. Create projects concerning the real equipment, and create associations between them and the existing standards and rules.
4. Simple to use approval method (with simple web browser access).
5. Keep a record of all performed operations.

1.3.2 Functional Requirements

1. Should use the already existing user data base for authentication.
2. Should have a role and permission system, where each user has only one role.
3. Multiple users should be allowed to work simultaneously, but if they try to work on the same project, they should be notified.
4. Support for creation and editing of new standards.
5. Should allow to insert or edit topics and rules for each standard.
6. Each rule should be allowed to contain text information and images.
7. Should allow for standards and the corresponding topics to be printed.
8. Should allow for projects to be created, edited and viewed by the users using User Access permissions.
9. The users should be able to create equipment on the respective projects.
10. After the equipment is created, rules should be able to be assigned to them.
11. Should have an approval mode where the users can indicate if the rules are being followed by the physical equipment or not.

12. On the approval mode, the following information about the assignment of a rule to an equipment must be displayed:
    - Status (Open or Closed)
    - Comment on the last visit
    - Responsible for the last visit
    - Responsible for fixing the problem
    - Due date for the next visit

1.3.3 Non functional requirements

1. The application must be developed using the Blazor framework, Web assembly mode, and .Net 6.
2. The application must use Entity Framework Core to interact with the database.
3. The application must use the JSON Web Token standard for the session management of the system.
4. Each project must contain a number, a description, and responsible users.

1.4 Current Practice

The current method being used by the company consists on having the standard and rules information stored and described on the already mentioned PowerPoint files, and using an Excel spreadsheet to keep a track of the needed and approved rules.

When a user has to perform an approval operation on equipment, he has to do the preparation work of summarising the needed rules and add them to the spreadsheet. When the time for approval comes, the user will just note down the result on that Excel file.

The main disadvantages of this method can be found on all the preparation work that is needed on creating the approval spreadsheet, where only a short version of the rules is described, if the need for more detail arises, the worker has to search for it on the original file. Since several users perform different approvals to different projects and equipment, the result would be a large number of files, which could be hard to keep track of.

1.5 Goals

The goal is to create a functional platform to be deployed and used by the workers at Preh Portugal by the end of this dissertation.

On the first deployed version the application should be able to perform the following operations:

- Allow the users to access and manage the standards stored on the database.
- Create and edit projects, and manage all equipment present on them.
- Approve the rules assigned to equipment on the dedicated approval mode.
- Assign new rules to existing equipment on the dedicated management mode.
- Have a record of all approval operations that can be verified when needed.
- Print relevant information from the standards.
- Print relevant information about projects and equipment.

From the multiple existing standards, the electrical one will be the first one to be adapted and implemented in to the application. However, the app should be scalable to incorporate new standards while maintaining the format.

Chapter 2

State of the Art

2.1 The Company: Preh Portugal

Preh Portugal, located in the city of Trofa, is the Portuguese settlement of the German group Preh. This group specializes in producing automotive products like car HMI (human machine interfaces), commercial vehicle HMI and developing E-Mobility components.
The facility in Portugal has the main focus of producing electronic components for the industry described above.
At the industrialisation division new projects for producing and testing those components are built. Those projects need to be certified and approved following some regulated standards. Inside this division, exists the software development department, that is responsible for the development of those projects, and is also where the project of this dissertation is being conducted.

2.2 Market survey

Concerning already existing products, some solutions were found and considered feasible as a possible way to solve the current problem.

Filedoc: Filedoc is an integrated document and process management solution that encompasses archive functionalities, document life-cycle management and process management. The main goal of this application is for the management of documentation and processes to be done in an easy, fast and efficient way, allowing for a greater control and handling of all the information stored there.

RedHat: Consultancy company specialised in developing open source cloud solutions.

The options presented above can fulfill some of the requirements presented before, but there isn't one that can guarantee that all the requirements are met, and they also have added cost for the services that they offer. So developing a custom made application is the best solution to assure that the needed requirements are met, and to leave the idea board open for new features and requirements.

2.3 Scope

The decision of creating a new platform from scratch has both advantages and disadvantages, that are presented on the following sections.

2.3.1 Advantages

Since it is going to be custom developed, the platform will be focused on making sure that all the requirements are met, as well as all new requirements or new features that might appear can be implemented with ease, making this application expandable and scalable for more features and information. By being internally developed by the company, the implementation and adaptation of the application for the already existing hosting server and database easily done and controlled, allowing for modifications to the source code if needed.

2.3.2 Disadvantages

By being developed with the objective of responding to the requirements presented above, the application will have a small utility range, only being useful for the industrialization users that are going to be responsible for the approval operations, making it a system that has a very specific goal and usability.

After considering all these factors and the options found on the market survey, the conclusion was that creating a new and customized platform was the best decision to make.

2.4 Tools and Technologies

As previously stated, the desired result for this platform was to provide a better method for users to perform approval operations, including all the preparation work, the operations themselves and keeping a record of all previous performed operations. So far to perform this task the users have to go through a arduous preparation work of creating an excel spreadsheet with all the rules needed to evaluate, if they have any doubts during the process they have to consult the document containing all the information, and even after the operation is performed, it is difficult to keep track and a record of all the previously performed ones.

The two possible options for the creation of this platform are the following:

Desktop Application: Software that is installed locally on a computer. It can be launched whenever needed and independently of other applications. It takes up space from the hard drive and can work regardless of internet connection, although some need it to function as intended.

Web Application: Type of software application that is used through the internet via a web browser. Instead of storing the files on the used computer, it is located on a remote server. A web

browser allows for access the app and its content and also runs all the scripts responsible for its features. What differentiates a simple static web page from the web application is interactivity. They often allow for creation or manipulation of data and content.

To decide which one of these options is better, we had to analyse and weight the advantages and disadvantages of each software implementation.

Desktop Application

Advantages:

- Less need for internet connection
- Security
- Better performance
- Optimal use of computer resources
- No ongoing hosting costs

Disadvantages:

- Less portable
- Hard drive space
- Mandatory installation
- Needs to be manually updated
- Deployment for each PC
- Operative system compatibility issues

Web Application

Advantages:

- No need for installation
- Automatic updates
- Cross-Platform availability
- Mobile access
- Light on computer resources

Disadvantages:

- Security threats
- Dependence on internet access
- Potentially slower than desktop equivalents

Using the comparison table 2.1 and according to the characteristics presented above and the project requirements, we concluded that the most suitable option was to implement the platform as a web application. The company has already been implementing platforms as web applications, helping on the decision and development process.

Table 2.1: Comparison between the two considered application platforms

Desktop		Web
Not required	Internet connection	Required
Manually installed	Instalation and updates	No intallation with automatic updates
Platform-dependent	Platform dependency	Cross-platform and portable
Faster and with a wider range of features	Performance	Complex processes can be slower and dependant on internet connection
More secure	Security	Considered to be less secure

2.4.1 Web application structure

The two big parts of a web application can be considered to be the:

Client Side

Client side refers to everything in a web application that is displayed or takes place on the end user device. This includes what the user sees, such as text, images, and the rest of the user interface, along with any actions that an application performs within the user's browser. The tools used for this have remained roughly the same through the years, and are the pillars to build every web application.

HTML: HyperText Markup Language is responsible for structuring the content of a web page, making it one core technology of web development. HTML elements, that can be identified by the "<" and ">" characters that act as delimiters, describe to a web page how to display text, images, and all the other available features.

CSS: Cascading Style Scripts is a language for denoting the presentation of a web page. Colors, layouts, and fonts are some of the integral characteristics of a web page and application, that are described by CSS.

JavaScript: JavaScript is the principal client side programming language for any type of web development. Together with HTML and CSS finishes the core of technologies of web development, allowing for developers to build dynamic websites.

Server Side

Server side refers to everything that happens on the server, instead of on the client. In the past, nearly all business logic ran on the server side, and this included rendering dynamic web pages, interacting with databases and identity authentication. The problem with hosting all of these processes on the server side is that each request involving one of them has to travel all the way from the client to the server, every time. This introduces a great deal of latency. For

this reason, contemporary applications run more code on the client side, for example, rendering dynamic web pages in real time by running scripts within the browser that make changes to the content a user sees.

Programming Languages: Server side programming languages must handle the logic of a web application that take place behind the user interface. It involves working with databases to send and receive data from one end to the other, managing user connections and security authentications, and ensuring the web application performs as desired. Some of the most popular used languages are Java, Python, Ruby and PHP which use can be simplified by the use of frameworks to make the development process easier.

Databases: A database is an organized collection of data stored and accessed electronically. It can be used by a web application to store the data required for the required functionalities. Structured Query Language (SQL) is the go-to query language for the common web developer but it has many extensions or related versions extending additional functionality such as MySQL, PostgreSQL, Microsoft SQL, and Oracle.

Servers: Servers respond to network requests. Through the internet connection of a web application, a server retrieves information based on client requests and then serves the client. While there isn't a specific language for servers, all of the technologies that make up back-end development should absolutely have a good relationship with the servers they work with.

When it comes to choosing which technologies to use for either client and server side of the server application, there are a lot of options available, each one of them with their respective advantages and disadvantages. For previous applications and for other services the company has been using the tools offered by the Microsoft ecosystem. Considering this the .Net framework, version 6, was used for both sides of the web application. For storing all the needed data a Microsoft SQL Server was chosen and to interact with it, the Entity Framework was also chosen to be used.

HTTP

The two sides of the application can communicate using the Hypertext Transfer Protocol (HTTP), which is a protocol for transmitting hypermedia documents, such as HTML. It was designed for communication between web browsers and web servers, but it can also be used for other purposes. HTTP follows a classical client-server model, with a client opening a connection to make a request, then waiting until it receives a response. HTTP is a stateless protocol, meaning that the server does not keep any data (state) between two requests. For this web application the HTTP protocol will be used to transmit information using JSON format, containing the needed data requested from the server and information about the pages to display on the browser.

Having chosen the main tools to build the platform, all that was left was to tackle was the security aspect of the application when it comes to authentication and authorization to use it. This

is an important aspect for the web application since it is needed to prevent people that should not have access to the application from doing so. For the users to authenticate it was decided that they have to use their control number joined with the password of their choice. But there is still the need to keep the logged in user session live while they are still using the application. To do this, two main methods were considered:

Session or Cookies approach: After a successful authentication the server generates a "sessionId" that is saved on a "sessionDb" and a cookie is sent with the Id to the browser, corresponding to the client side. The browser receives it, saves and stores it in the dedicated cookie storage. After that, every subsequent request to the server performed by the browser includes the cookie.

JSON Web Tokens (JWT) approach: After a successful authentication the server creates an "accessToken", encrypting the "userId" and expiration date parameters, and sends the generated token back to the browser that receives it and saves it on the client side. After that this "accessToken" is included in every request to the server.

Session or Cookies approach

Advantages:

- The logout operation is more precise since it occurs at the moment the "sessionId" is removed from the database

Disadvantages:

- Additional DB search operations are needed to get the userId from the sessionId table
- Individual servers cannot be scaled separately since they would need to share the sessionDB

JWT approach

Advantages:

- The "userId" parameter is obtained by decrypting the JWT token, no DB call is used to get it, making this a fast process
- Servers can be scaled separately, without the need share "sessionDB"
- The same token can be used to authenticate on different servers without the need to share "sessionDB"

Disadvantages:

- JWT tokens can only be invalidated at the end of their lifespan (typically 10 to 15 minutes)
- JWT tokens can contain a lot of irrelevant information. The tokens should only contain the necessary information for the authentication and authorization process, in order to keep them as short as possible.

Both methods were considered for the session management, and the JWT approach was the chosen one for this web application. This is due to having more advantages regarding scalability and because it has also been used on other company web applications.

2.4.2 .NET 6

The .NET Framework is an initiative of the Microsoft company, which aims at a single platform for development and execution of systems and applications. Any code generated for .NET can run on any device that has a framework for that platform. With similar idea to the Java platform, the programmer stops writing code for a specific system or device, and starts writing for the .NET platform. Applications written for it run in a controlled software environment, as opposed to a hardware environment, through an application virtual machine.

It is formed by the Framework Class Library (FCL) which is a large library of classes and is executed in a software environment named Common Language Runtime (CLR), which is an application virtual machine that provides services such as security, memory management, and exception handling.

The FCL provides the user interface, data access, database connectivity, cryptography, web application development, numeric algorithms, and network communications and language interpolation, allowing the for languages to use code written in other languages. Programmers create software by combining their source code with .NET Framework and other libraries.

ASP.NET is the open source web application framework that extends the .NET developer platform with tools and libraries for building web apps. It allows for the creation of dynamic web pages using any programming language supported by the .NET framework.

On this framework, there are included Razor pages, which is a part of a lightweight and flexible framework that provides full control over rendered HTML to the developer. It is a server-side page-focused framework that enables building dynamic, data-driven web applications. It makes use of the popular C# programming language for server side programming, and the easy to learn Razor templating syntax for embedding C# in HTML, to generate content for browsers dynamically, figure 2.1.

```
@page "/counter"

<PageTitle>Counter</PageTitle>

<h1>Counter</h1>

<p role="status">Current count: @currentCount</p>

<button class="btn btn-primary" @onclick="IncrementCount">Click me</button>

@code {
    private int currentCount = 0;

    private void IncrementCount()
    {
        currentCount++;
    }
}
```

Figure 2.1: Razor page example

ASP.NET offers a powerful, patterns-based way to build dynamic websites using the MVC pattern that enables a clean separation of concerns.

MVC

The Model-View-Controller (MVC) [1], architectural pattern separates an application into three main groups of components: Models, Views, and Controllers. This pattern helps to achieve separation of concerns. Using this pattern, user requests are routed to a Controller which is responsible for working with the Model to perform user actions and/or retrieve results of queries. The Controller chooses the View to display to the user, and provides it with any Model data it requires. The interaction of all this parts can be seen on figure 2.2.

Model Responsibilities: The Model in an MVC application represents the state of the application and any business logic or operations that should be performed by it. Business logic should be encapsulated in the model, along with any implementation logic for persisting the state of the application.

View Responsibilities: Views are responsible for presenting content through the user interface. They use the Razor view engine to embed .NET code in HTML markup. There should be minimal logic within views, and any logic in them should relate to presenting content

Controller Responsibilities: Controllers are the components that handle user interaction, work with the model, and ultimately select a view to render. In an MVC application, the view only displays information; the controller handles and responds to user input and interaction. In the MVC pattern, the controller is the initial entry point, and is responsible for selecting which model types to work with and which view to render.

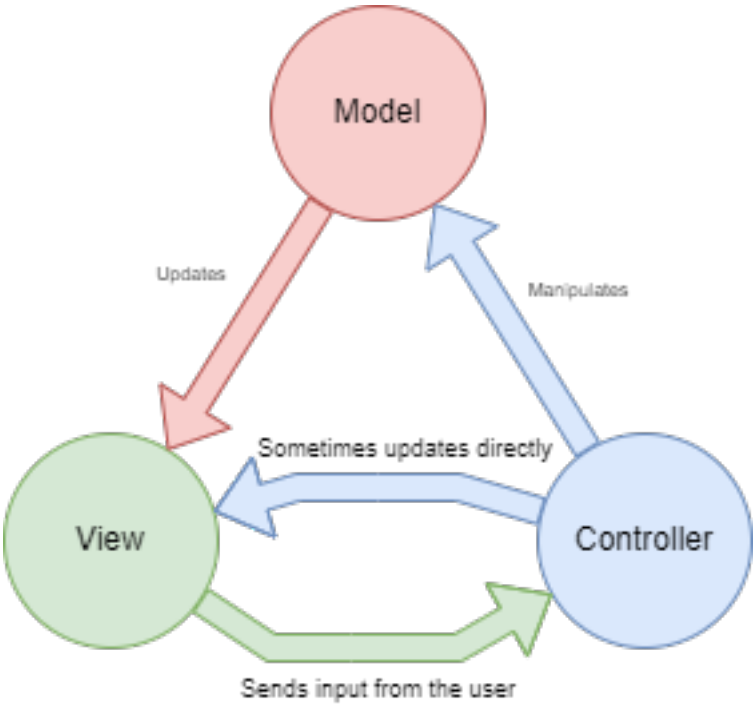


Figure 2.2: MVC diagram

On the ASP.NET environment, there is a feature called Blazor that allows for the development of Single Page Applications (SPA). As the name states, SPA are web applications or websites that interact with the user by dynamically rewriting the current web page with new data from the web server, instead of the usual method of a web browser loading entire pages. This approach results in faster transitions, making the website feel like a native application. A page refresh never occurs, and instead, all the necessary content is dynamically loaded to the page as necessary, usually in response to user actions. This feature was chosen as the main method for the development of the web application.

2.4.3 Blazor

The name Blazor is a combination of the words Browser and Razor (the .NET HTML view generating engine) [2].

This framework was created by Microsoft and enables developers to build single page applications using C#, instead of the traditional JavaScript. Both the client and server code are built using C# allowing code and libraries to be shared between them.

Blazor can be hosted in two distinct ways:

WebAssembly (WASM): Runs the client-side C# code directly in the browser, using WebAssembly. Because it's real .NET running on WebAssembly, code and libraries from server-side parts of the application can be re-used.

Server: Runs the client logic on the server. Client UI events are sent back to the server using SignalR - a real-time messaging framework. Once execution completes, the required UI changes are sent to the client.

WebAssembly

Advantages:

- Faster UI Code
- Higher performance
- Offline support
- Any .NET standard 2.0 C# can be run

Disadvantages:

- An API layer is required if access to secure resources is needed
- Debugging still a bit limited

Server

Advantages:

- Faster loading than WASM
- Easy access to secure resources like databases or cloud-based services
- Can be used by browsers that don't support WASM
- C# code is not sent to the browser

Disadvantages:

- Extra latency due to sending data back and forth
- No offline support
- Scalability can be challenging
- Server required

Since this platform is going to be used by a large numbers of users and good performance is a required feature, Blazor WebAssembly was chosen.

2.4.3.1 Blazor WebAssembly

According to the author in *WebAssembly for Cloud* [3], WebAssembly (WASM) is a safe, portable virtual instruction set designed to run on any host capable of interpreting those instructions.

It is formatted in a specific binary format. Any host (hardware or software) that adheres to this specification is therefore capable of reading binaries and executing them – either interpreted, or by compiling directly to machine language specific to the device.

This new binary format, which is optimized for browser execution, allows for a wide range of programming languages that can be compiled into this format to run on the browser, since all major used browsers support this feature, including their mobile versions.

Mono is the open source implementation of the .NET CLI specification, meaning that it is a platform for running .NET assemblies. The approach used by Blazor, consists on compiling

the Mono runtime into WASM, allowing it to run on the browser and execute .NET Intermediate Language just like regular .NET apps.

As previouly said, Blazor is used to create interactive client-side web apps using .NET. The WebAssembly version of Blazor, uses open web standards without plugins or recompiling code into other languages, resulting on all modern web browsers being able to support it, including mobile browsers.

When a Blazor WebAssembly application is built and run on a browser:

- The C# and Razor files are compiled into .NET assemblies.
- The assemblies and the .NET runtime are downloaded to the browser.
- Blazor WebAssembly bootstraps the .NET runtime and configures the runtime to load the assemblies for the web application. The Blazor WebAssembly runtime uses JavaScript interop to handle DOM manipulation and browser API calls.

2.4.3.2 Components

Blazor applications are based on components. A component is an element of user interface, such as a page, dialog, or data entry form.

Components are classes that belong to the .NET platform that have the following properties:

- Can define flexible UI rendering logic.
- Allow the handling user events.
- Can be nested and reused.

These component classes are written in the form of a Razor markup page, ending with a .razor file extension. This format allows the developer to switch between HTML and C# within the same file.

The razor files are then executed inside the browser to dynamically build a web page. This eliminates the need to use JavaScript, even though it can be included and used.

2.4.4 JSON Web Tokens

JWT is an open standard that defines a compact and self-contained way for securely transmitting information between parties as a JSON object. JWTs have the option to be digitally signed using a secret encrypted key, making this a secure, trusted and verified method to store and share information. One of the most common use for these tokens is authentication and authorization [4]. Once the user is logged in, each subsequent request will include the JWT, allowing the user to access routes, services and resources that are permitted with that token. Using this method the session information is stored on the users browser instead of the server.

This standard defines claims that correspond to a statement about an entity, either the user or the token itself, and additional data.

There are three types of claims:

Registered claims: Set of predefined claims about the token that are not mandatory, but recommended to provide a set of useful and interoperable claims.

Public claims: Claims that can be defined at will by the developers. To avoid name collisions, they should be defined according to the *IANA JSON Web Token Registry* [4].

Private claims: Custom claims created to share information between parties that agree on using them and that are not registered or public claims.

2.4.4.1 Structure

In the most simple form, JSON Web Tokens consist of three parts separated by dots. Those parts are:

- Header
- Payload
- Signature/Encryption data

Header

Every token carries a header with claims about itself regarding the type of token it is, and which encryption algorithm is used for the signature. An example of a simple header can be found bellow:

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

Payload

The payload is the element where all the interesting user data is added using claims. There are no mandatory claims that need to be present on the payload, but there are some recommended ones since they provide useful information. These recommended are:

iss: (issuer) defines the party that issued the JWT.

sub: (subject) identifies the party that the token carries information about.

aud: (audience) identifies the intended recipients of the JWT.

exp: (expiration time) a number representing a specific date and time. This claim sets the exact moment from which the token is considered invalid.

An example payload could be:

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "admin": true
}
```

Signature

The signature is used to verify if the message was not changed along the way, and if the token is signed with a private key, it can also verify that the sender of the JWT is who it says it is.

To create the signature, all that is needed is the header and payload encoded in an Base64-URL format, and a secret. Afer all the parts are joined, they can be encripted using the algorithm chosen and defined in the header. On the following example, a HMAC SHA256 algorithm is used:

```
{
  HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret)
}
```

Joining all parts

By joining all three previous parts, the result is three Base64-URL strings separated by dots, that can be easily parsed.

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4
gRG9lIiwiaXNTb2NpYWwiOnRydWV9.
4pcPyMD09olPSyXnrXCjTwXyr4BsezdI1AVTmud2fU4
```

Figure 2.3: JWT example

2.4.4.2 How do JWTs work for authentication?

When the user successfully logs in, a token containing all the respective information stored in claims will be returned from the server and stored on the browsers storage. Whenever the user performs a request to access a resource on the server side, the JWT should be included on in the authorization header using the *Bearer* schema. The server's protected resources will check for a valid token in the request header, and if present, the user will be allowed to access them.

2.4.5 Data Base

For the Web application to be dynamic and to achieve the desired functionalities, a method to store and manage data was required. The best way to accomplish this is through the use of a database system and there are a lot of different methods when it comes to implementing it. On the Microsoft environment the main option to solve this need is the Microsoft SQL Server. This tool allowed for the creation and maintenance of relational databases making it suitable for this project.

During the development and testing faze, a DB hosted on the local machine was used to allow for more freedom with testing and management of data.

The database structure was designed using a UML class diagram [5] to identify and clarify the relations between the tables that would form the database. This diagram went through several changes though time in order to keep up with all the features and to be approved by the product owner. The final class diagram can be found on figure 2.4:

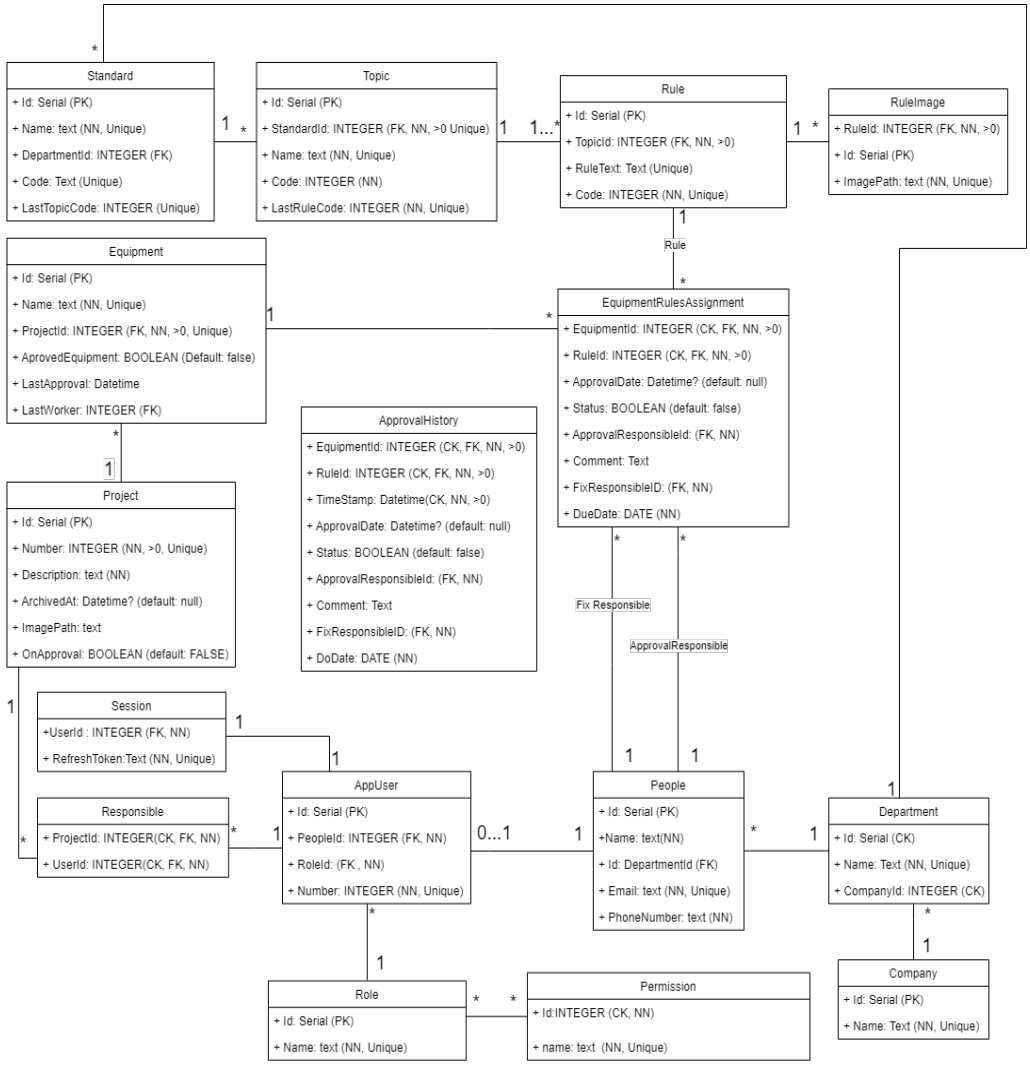


Figure 2.4: Used Data Model

The classes composing this diagram can be considered as three different parts, each one of them concerning a different part of the application.

Documentation: Classes where all data regarding standards and the information that composes them is stored.

Project and Approval: Classes where the projects and the respective equipment data is stored. All approval operations records are also kept here.

User data: Classes that store all the information required regarding the users that have access to the application.

Documentation:

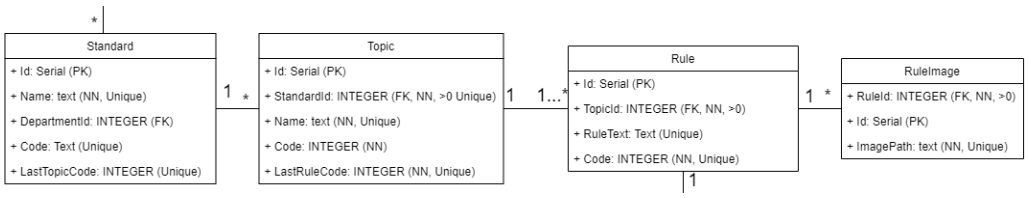


Figure 2.5: Documentation classes

The four classes on the top part of the diagram, figure 2.5, form the documentation classes where the information about the standards are stored.

The "Standard" class represents the various existing standards. It is formed by the index that is used as a primary key, a unique name that cannot be **null**, the ID of the department which the standard belongs to, the code attribute that is a unique identifier for that standard and the code of the last topic, representing the code index from the last topic that is assigned to the standard. The "Topic" class has similar attributes, the unique identifier as the primary key, the StandardId as a foreign key to identify the standard it belongs to, the name for the topic as an unique string, the code attribute that now is an numerical value and works as an index for each topic within that standard and the last rule code, that saves the last index of a rule. On the "Rule" class, the properties consist on the identifier as primary key, the TopicId as a foreign key identifying the topic they are assigned to, the text defining the rule and a numerical unique code working as an index for that rule. The last class is the "RuleImage" on. This class makes the connection between the path to each image and the rule they belong to. The class is composed by the serial identifier as a primary key, the RuleId to identify the rule to where the image is assigned and in the end an unique and mandatory path to the respective image file.

The relations between these classes are basically the same, they are all one to many relations. This is due to a Standard being composed of Topics that are composed of Rules, and each rule can have a number of descriptive images.

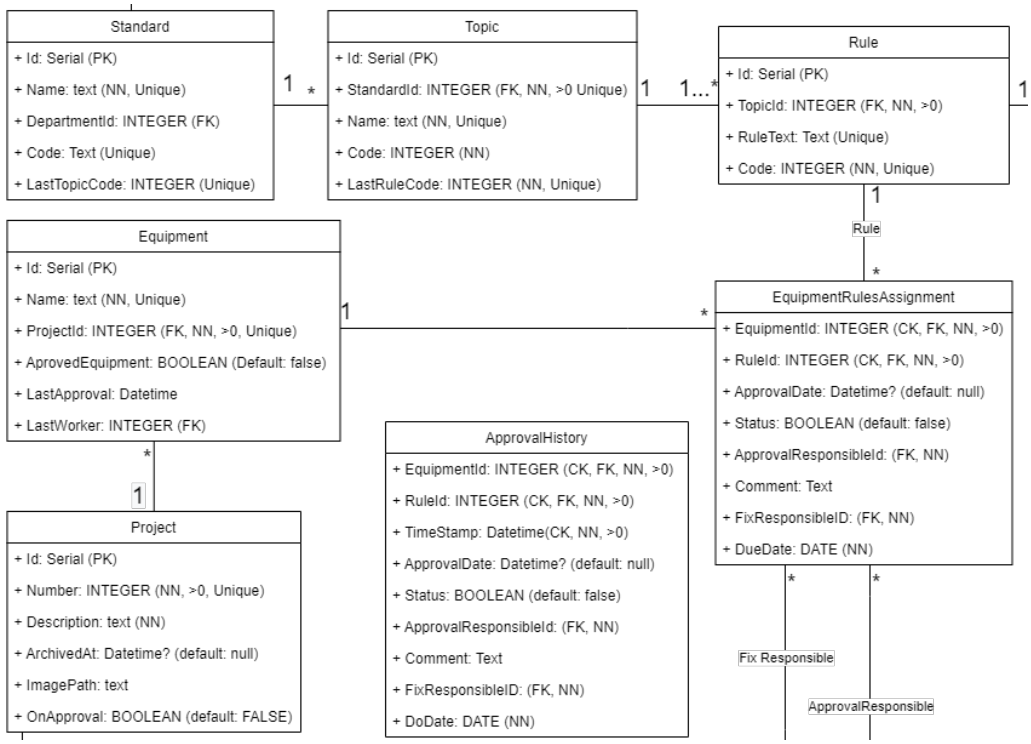


Figure 2.6: Project and Approval classes

Project and Approval:

The four classes on the middle, figure 2.6, are the ones where the information about the projects equipment and all the performed operations is stored. Starting on the "Project" class, it is formed by the index attribute for a primary key, and mandatory and positive number that is unique for each project, a short text description, a datetime attribute to represent the timestamp when the project was archived, if **null** that means the project is not archived, a string representing the image path pointing to the image specific to that project, if this parameter has a **null** value a default image is used by the application and to finish a boolean attribute representing if the approval process on that project already started. Each project can be composed of several equipment, but a single equipment can only be present on one project, so there is a relation of one to many between the classes. The "equipment" class is formed by the Id attribute for the primary key, a mandatory and unique name, the ProjectId as a foreign key identifying the project it belongs to, a boolean variable to identify if an equipment is already approved or not and two attributes to identify the last worker that loaded or changed data of this class and when it happened. The "EquipmentRulesAssignment" class is the one that creates the link between the equipment and each one of the rules assigned to it. It has a compound key formed by two foreign keys that point to the respective equipment and rule that assignment is linking, a datetime attribute representing when the last approval operation was performed, a boolean attribute named status to identify if that assignment has been approved or rejected, a foreign key to the people table identifying who was the responsible for the last performed operation, the comment left on the last performed operation

and a date attribute to provide a due date until when the next operation has to be performed. The last class "ApprovalHIstory" is a duplicate of the last mentioned class and is used to keep a record of all performed operations and only has an additional attribute, the timestamp when the operation was performed.

User data:

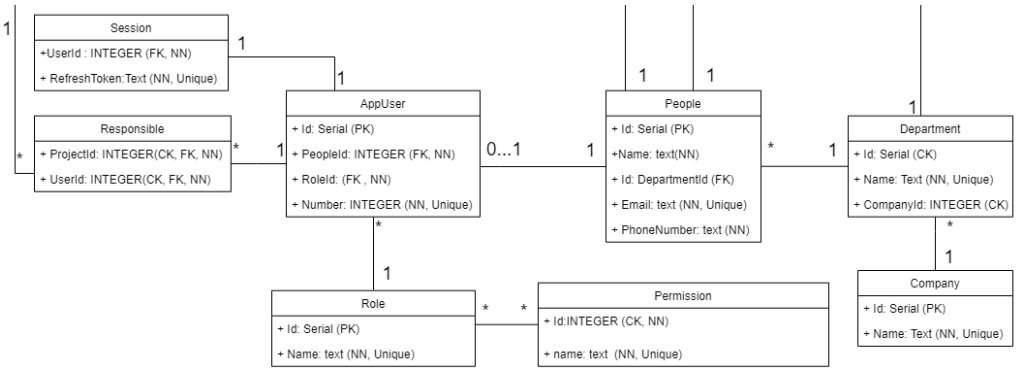


Figure 2.7: User data classes

The bottom classes of the diagram represent the classes are going to deal with the user and session data for the application. The "People" class saves the information about each person that is needed by the application. It has the serial Id, the name of the person, a mandatory and unique email and the phone number. Each person must be assigned to a department and that department must be a part of a company, resulting in having one to many connections between them. The class "AppUser" represents all the users that have access and permissions to use the application. It is formed by the serial Id, the control number specific to that company user, and two foreign keys, one identifying the person entity of that user on the "People" table and another foreign key to identify the role that the user has. Each user has a session class specific to it, in order to keep the authentication state. The "Responsible" class makes the connection between each users are responsibles for each project. It only has two attributes, the ProjectId and the UserId, forming a compound key. The "Role" class, composed by the serial Id and a unique name, has a relation of one to many with the user class, since a role can be present in several users, and an user can only have one role. The "Permission" class, formed by an Id and a name, can be present in many roles, which can have several permissions, so a relation of many to many was assigned between these two classes. Using this permission and role method, access to different functionalities and pages of the application can be granted access or denied based on the permissions granted to each role.

2.4.6 Entity Framework

Interacting and performing operations on the database is an important need in a web application. It can be performed through manager apps, like the SQL Server Management Studio from the Microsoft environment, that needs to be configured to the correct DB and is only suitable for

humans to interact with it, or using a structured query language. Using and writing SQL code can be a time consuming and hard task for developers. To simplify this aspect and avoid writing SQL code directly, the Entity Framework was used to perform all operations regarding the database.

Entity Framework (EF) is an open-source Object-relational Mapping (ORM) framework for .NET applications. This framework allows for developers to interact and work with the database data using object domain specific classes without focusing on the underlying database tables and columns where the data is stored. This allows for a higher level of abstraction when dealing with data, as well as creating and maintaining data-oriented applications with less code, compared to traditional applications [6]. The use of this framework allowed for a simpler method to create and populate the initial database and to manage data of the web application over passing the need to write SQL code during the development.

Figure 2.8 demonstrates how Entity Frameworks fits into an application.

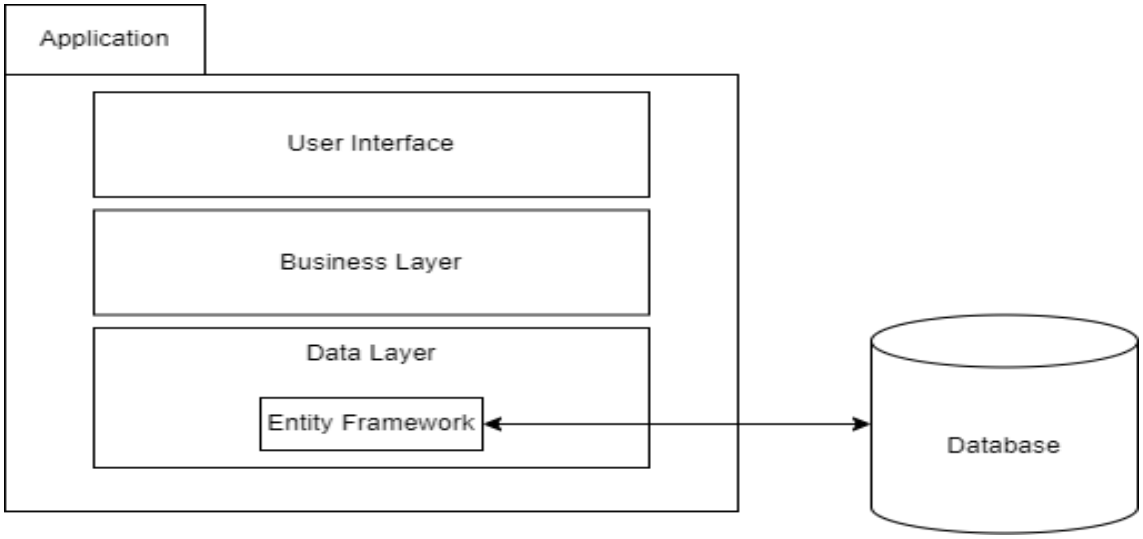


Figure 2.8: Entity Framework layer diagram

2.4.6.1 Context Class

The context class is the main and most important class while working with Entity Framework. It represents a session with the desired database, and is through this session that create, read, update and delete (CRUD) operations can be performed.

This class is derived from the already existing "DbContextDbContext" class, and is the one used to query or save data to the database, as well as configure domain classes, database related mappings, change tracking settings, etc.

2.4.6.2 Entity

In Entity Framework, an entity is a class that maps to a database table. This class must be included in the DbContext class as a DbSet<Entity> for it to be correctly considered as a database

table. EF API maps each entity to a table and each property of an entity class to a column in the database.

An entity can have two types of properties:

- Scalar Properties
- Navigation Properties

2.4.6.2.1 Scalar Properties

Scalar Properties are considered the primitive type properties, which are mapped to a column in the database table to store actual data. An example for this property is the unique ID identifiers typically used as primary keys. EF API creates a column in the database for each scalar property.

2.4.6.2.2 Navigation Properties

The Navigation Property represents a relationship to another entity, which also have two different types: Reference and Collection

A reference navigation property points to a single entity, representing a multiplicity relation on one in the entity relationship. When processing this property, EF API creates a foreign Key column that references a primary Key from another table in the database.

When an entity includes a generic collection of an entity type, it is considered a collection navigation property, and represents a multiplicity of many. EF won´t create any column for the collection, creating instead a column in the table of an entity of that generic collection.

2.4.6.3 Migrations

Having control of the structure of a database is one important aspect that should not be discarded when they are utilized. The first step is to create the data model that is going to be applied to it, than that model used to create a schema on the database. In order to create it, either SQL is used or any database manager software that offers the users the functionality to do it manually.

Entity frameworks has the migrations system that is a method to keep the database schema in sync with the EF model by preserving data. This means that this framework can alter the structure of database to replicate the existing domain classes. This way the schema can be built and replicated to other databases using only the files containing the migrations, overcoming the need to write SQL code directly. It comes with the cost that for any modification needed to be performed on the database, a new migration has to be created.

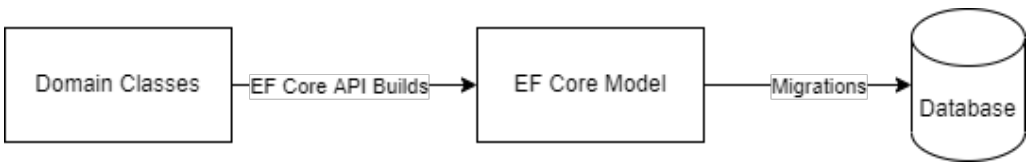


Figure 2.9: Migration Diagram

As shown in figure 2.9, after the domain (entity) classes are built, the EF API builds the respective model and creates a migration to apply on the database. These migrations contain all the information regarding the creation of tables, the relations and constraints between the columns. Whenever the domain classes are changed, a new migration has to be built and applied to the database, in order for these changes to take effect on the database schema.

On a regular migration file, there are two main sections defined. One that is executed when the migration is applied to the database, defined as the Up function, and one that can be used to revert the changes applied by the previous section, defined as the down function, figure 2.10.

```
public partial class Initial : Migration
{
    protected override void Up(MigrationBuilder migrationBuilder)
    {
        migrationBuilder.CreateTable(
            name: "Companies",
            columns: table => new
            {
                Id = table.Column<int>(type: "int", nullable: false)
                    .Annotation("SqlServer:Identity", "1, 1"),
                Name = table.Column<string>(type: "nvarchar(100)", maxLength: 100, nullable: false)
            },
            constraints: table =>
            {
                table.PrimaryKey("PK_Companies", x => x.Id);
            });
```

(a) Up function example

```
protected override void Down(MigrationBuilder migrationBuilder)
{
    migrationBuilder.DropTable(
        name: "Images");

    migrationBuilder.DropTable(
        name: "Permissions");
```

(b) Down function example

Figure 2.10: Migration functions example

Entity Framework keeps a file and a database table with a registry log from all the previous executed migrations. This way, it is possible to revert to previous database schemas easily. However, if this changes involve removing columns or tables, the data from that structure will be loss, with no possibility for retrieval.

2.4.6.4 EntityStates

During each entity lifetime, their respective state is maintained by the EF API. Each entity has one of the following states based on the operation performed on it via the context class:

1. Added - The context is tracking the entity but it does not exist on the database.
2. Modified - The context is tracking the entity, it exists on the database and its properties values have been altered.
3. Deleted - The context is tracking the entity and it exists on the database, but it has been marked for deletion.
4. Unchanged - The context is tracking the entity, it exists on the database and its properties values have not been altered.
5. Detached - The context is not tracking this entity.

Through change tracking, the context class holds the reference to all entity objects as soon as they are retrieved by the database, keeps track of their state and maintains modifications made to the properties of the entity.

The API builds and executes the INSERT, UPDATE or DELETE commands based on the entity state when the SaveChanges() method from the context class is called according to figure 2.11.

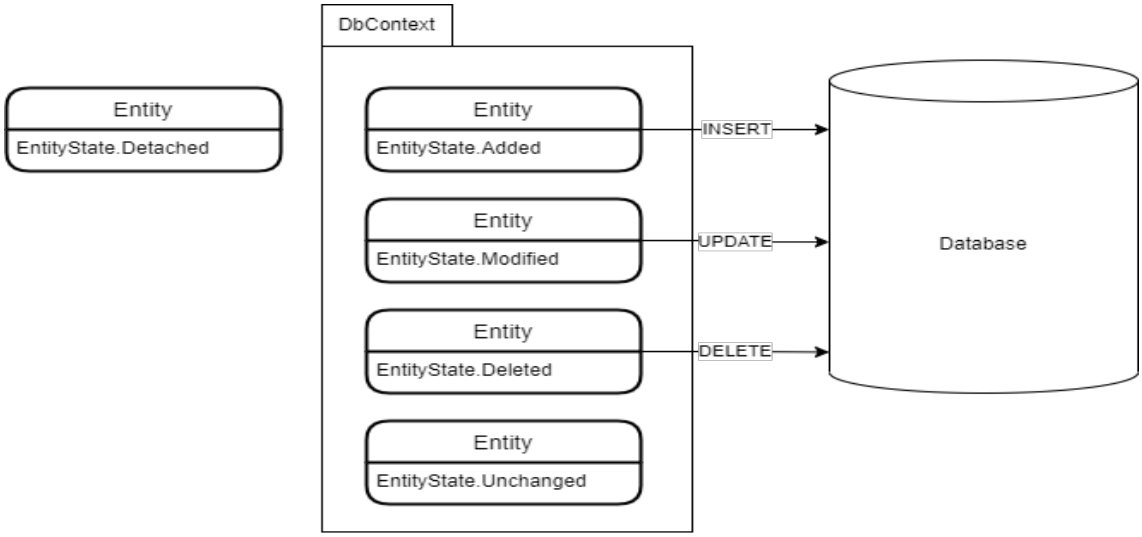


Figure 2.11: State influence on the database actions

2.5 User Interface and Experience

In order for an easy use and adaptability to this new platform, a pleasant user interface that provides a good experience while using it is an important aspect for the project.

The user Interface (UI) consists on all the elements that are displayed for the user to interact with them. It should provide all the necessary elements to grant the user the possibilities to access all the desired functionalities in an effective and natural way.

User Experience (UX), as the name states, refers to the experience that the user gets when interacting with the application. To offer the users the best UX the app should be responsive, intuitive and organized in order to make it easy to use.

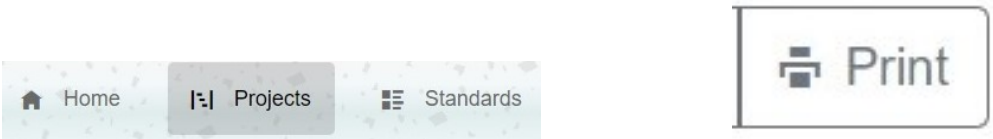
To achieve this, the most used method is combining HTML, CSS and JS to customize or create new elements that are the base structure of the web pages. Creating or adapting the existing components into having an appealing appearance is a time consuming and difficult task. To solve this problem several frameworks with already built components exist that can be included and their components used easily during the development. One of those frameworks is Bootstrap and it is already included into Blazor framework.

2.5.1 Bootstrap

Bootstrap is a free and open-source front end development framework for the creation of websites and web applications. It contains HTML, CSS and JavaScript design templates for several interface components. It offers a CSS library filled with several pre built components ready to be used during development. These components are joined by a JavaScript functions allowing for some of them to be more dynamic and reactive allowing for a wider range of option when choosing what element to use.

2.5.2 Open Iconic

To build a simple and clean interface, some type of icons have to be used, in order to have more intuitive elements, such as buttons or attention grabber information. Open Iconic is an open source and free to use icon set that can easily be integrated with the rest of bootstrap elements and that also comes integrated with the Blazor framework, making it the perfect choice for providing them. The large variety of icons combined with the simplicity of them and the fact that it was already included on the Blazor framework made this tool the perfect fit to include icons on the design of the web application pages.



(a) Navigation buttons (b) Print button

Figure 2.12: Page elements using icons

Chapter 3

Development

This chapter describes the development process of the project associated with this dissertation, from the implemented architecture to the developed features and the problems encountered through it.

3.1 Architecture

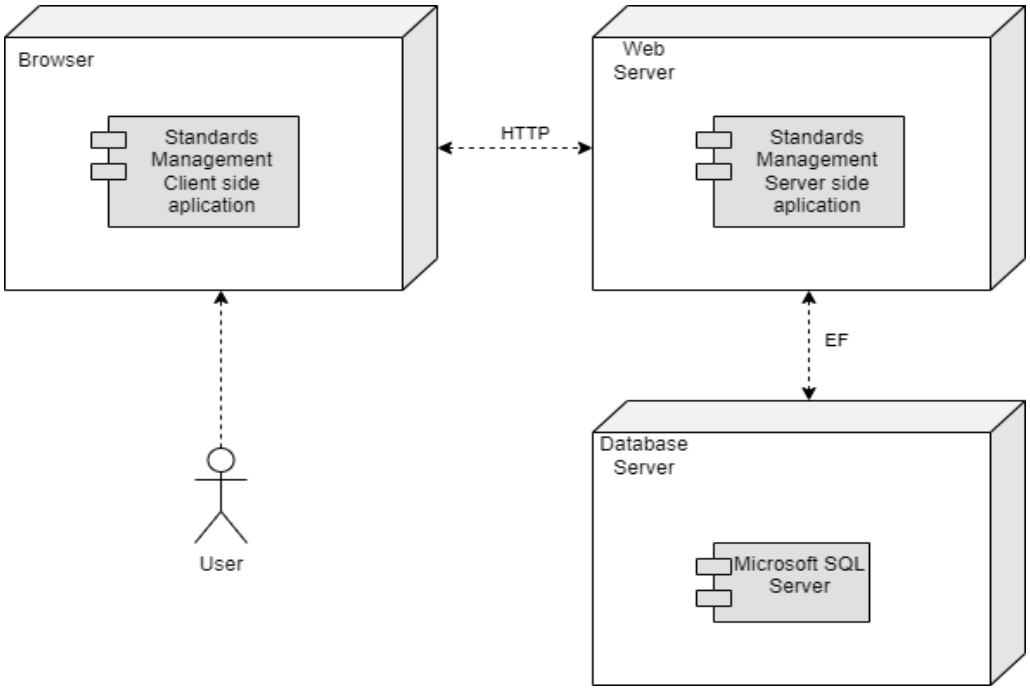


Figure 3.1: Deployment diagram

As shown on figure 3.1 the application is composed by 3 modules. The client side that is accessed by the user through an internet browser, the server side of the application that is being hosted on the company server and the database that is integrated on the company database server.

The users interact with the client side though the UI provided by the browser. The client side can request information from the server side through HTTP requests, that will then interact with the database using the EF context to perform the needed operations. The information regarding data that is sent back to the client as a HTTP response is sent the JSON format.

During the development, the project was divided into three main folders, client, server and shared, each one dedicated to different parts of the application. The client and server ones correspond to their respective sides on represented on figure 3.1 whereas the shared folder contained classes and other files that need to be accessed by both sides of the application, for example, this is where the classes needed for EF to work were defined.

3.2 Features Development

Creating the local data base

The first step into making a functional developing environment was creating the classes for EF to build the database tables and populating them. These are C# classes, where each property corresponds to a column on the table. To better identify the properties of each column, data annotations where used to clarify how each columns should behave.

After each class was created, they were declared on the EF context class. With all tables classes created and declared on the context class, the first migration was created and applied to the database.

Having the local database created, the next stage was creating and configuring with the correct properties the three project environments:

- Development
- Staging
- Production

Standards Management

The first set of features of the application to be the main focus of development were the ones related to the standards storage and management. This section was the best one to begin with, since it is the core part where all the information is going to be created and stored for the other segments of the application, and it was a good first challenge concerning getting used to the framework.

The first pages to be developed were the "create/edit standard", and the "list standards" pages. It started with creating the API functions for the server part of the application. These functions are the ones that are called through HTTP requests by the client, and that perform all the interactions with the database. After having the controllers available, the requests to access them were executed through service class functions, called by the razor pages defining the application pages.

For creating and editing the standards a single page was created, that has different behaviors depending on the desired functionality. Is is composed by an HTML form allowing to edit the

properties of an instance of the standard class to be edited. For displaying all the standards, after the response is obtained from the service, they are organized on a table, where each row is clickable, and to access more information about the standard, the user has to click on them.

On the page displaying all the information about one standard, all the topics and rules are displayed. When the page is initialized, a request is sent for the standard that was selected to display, and the response contains all the information concerning it. For this display page, the idea was to have the rules grouped by the topic they belong to, and have a user friendly way to access every piece of information. The first and final idea implemented was a bootstrap accordion. This component provided a way to hide or display the rules belonging to each topic in a simple and user friendly way 3.2.

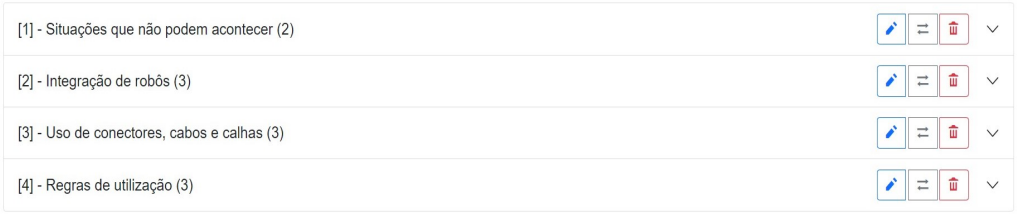


Figure 3.2: Topics Accordions

To create a topic, the only needed information is the name it should have. So for creating a topic, instead of having a new page, a text input and a button was used for that action, as well as for editing it.

Regarding creating the rules, the main goal was to have the user only adding the text and images needed for that rule, so once again the page to create and edit a rule is the same, and it is formed by a text area input and a file submission system for the images.

Project Organization

After having the standards part functional, the next big feature of the application, was the projects and equipment organization, together with their approval.

The first page to be developed was the one listing all the projects. Once again it started with the creation of the service functions for the client side, and the controllers for the server side of the application, in order for the information on be DB to be accessible.

Once again, the creation and editing of projects takes place on the same page, but it has different functionalities depending on the desired operations. When creating a product, a forms is present with a text inputs are displayed for the numerical code input and the project description. Together with that, a multi select option is displayed to chose the users that are going to be responsible for that project and a file input option to add an image for the project. For the editing operation, the only difference is that a toggle button to archive the project is displayed next to the submit button.

Since the projects are only described using a numerical code, a description and an image, the chosen display method for them was a bootstrap card 3.3.

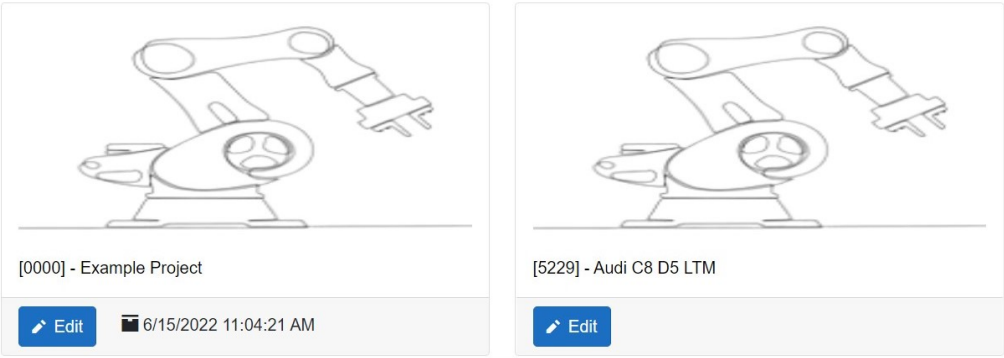


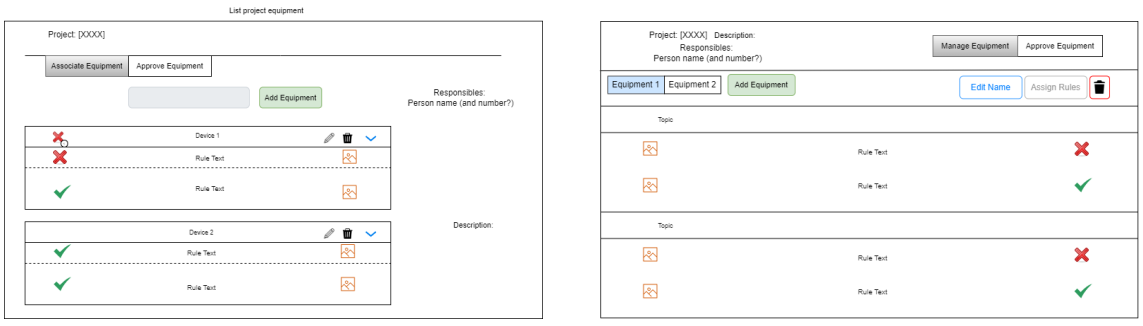
Figure 3.3: Project cards

These cards were made clickable, allowing to access all the equipment present in that project. The equipment display page was required to have to modes:

- **Management mode:** Used for creating and editing equipment, as well as assigning rules to them. Lists all rules with display purposes only.
- **Approval mode:** Used for approving rules that are present on each equipment. The rules are all listed with the intention of creating a checklist.

Since this page has different information to display depending on the selected mode, the common elements such as the project number, description, responsibles and the toggle buttons where defined on the main page, and both modes are built on two new components, that are displayed depending on the state of the toggle button. These two modes are:

Management mode



(a) Manage mode 1st mockup

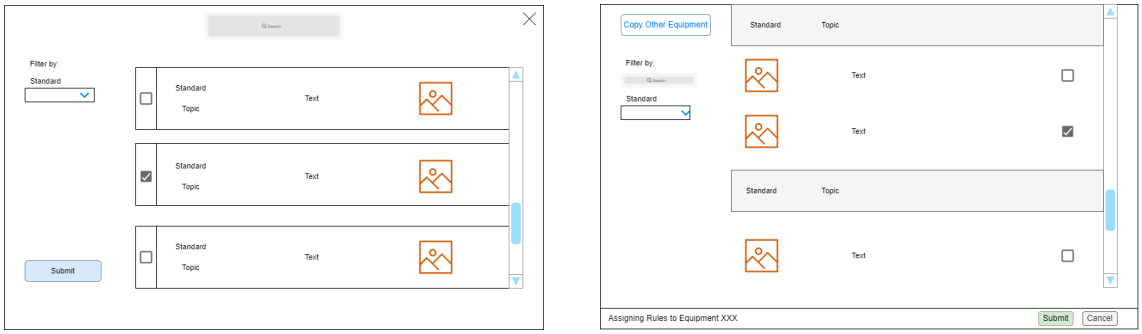
(b) Manage mode final mockup

Figure 3.4: Management mode mockups. Implementation: 5.12

As previously stated this mode is the main tool when it comes to create or edit the equipment, and then relate the corresponding rules with them. The controllers and services needed for these functionality mainly interact with the "Equipment" table, and the "EquipmentRulesAssignments".

After the addition of these functions, it was time to start developing the component. There are clear differences from the first mockup 3.4a and the final one 3.4b. The functionalities are basically the same, but it was adapted in order to keep a more concise design for the various modes.

All the equipment present on the project are aligned and displayed, on a menu bellow the project information header. Every equipment name is implemented as a button, to allow for a the user to quickly select different equipment. To create a new equipment for this, all that is needed is the desired name, so to add a new one, a single text input was used. This input can be accessed using the "Add Equipment" button. Too manage the selected equipment three simple buttons were added and implemented. The edit name and delete buttons execute the corresponding functions derived by their name. The "Assign Rules" button performs the job of enabling the popup window to assign rules to appear.



(a) Assign Rules 1st mockup

(b) Assign Rules final mockup

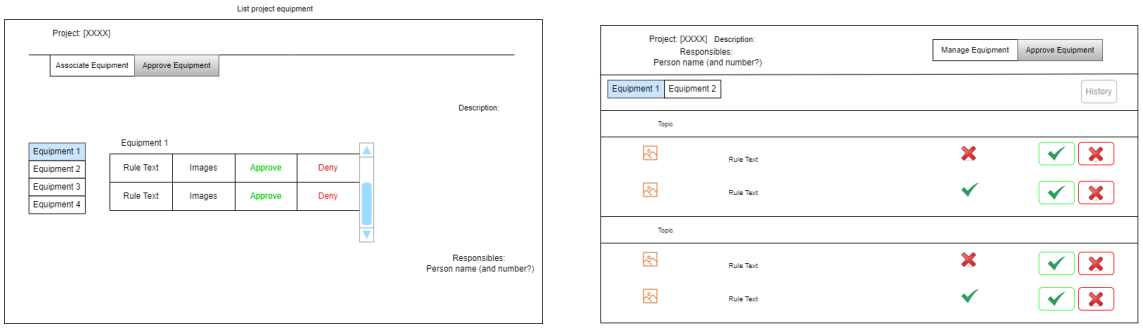
Figure 3.5: Assign Rules mockups. Implementation: 5.11

To create this appearing menu, the bootstrap modal component was used and adapted to implement the mockup present on figure 3.5b. The way this feature was implemented was listing all the rules present on each standard, and grouped by topic. Each rule has a checkbox, that when checked represents that the respective rule is assigned to that equipment. When the submit button is pressed, if the checked state of a rule was changed, the data on the DB table "EquipmentRulesAssignment" is modified according to the changes made. For checked rules new entries are added if they didn't exist before, and for unchecked rules, if an entry exists on the DB, it is deleted.

After having equipment with assigned rules, the method to display them was implemented following the designed mockup on figure 3.4b. The rules of the selected equipment are listed and grouped by topic. Each rule is described by the corresponding images, that can be clicked to open a popup window with a bigger description of the rule, the corresponding text and the current state of the rule, approved, rejected or without any performed operations yet.

Approval mode

After having the possibility to replicate the equipment on the application, the approval mode was the focus of development.



(a) Approval mode 1st mockup (b) Approval mode final mockup

Figure 3.6: Approval mode mockups. Implemetation: 5.13

The equipment listing and selection menu is the same as on the manage mode, and the selected equipment is kept selected while switching between the modes.

The creation of the checklist followed the mockup present on figure 3.6b. The rules are described the same way as they were for the previous mode, with the addition of the buttons to approve or deny the said rule.

(a) Approve rule confirmation mockup (b) Reject rule confirmation mockup

Figure 3.7: Confirmation popup mockups. Implementation: 5.14

To confirm the performed actions, two popup windows 3.7 were created depending on which action is chosen. For approving a rule, all the user only has the option to add a comment that can be optional. To reject a rule, the user has to add a mandatory comment, select a responsible for fixing that problem and a due date for that situation to be resolved. After the corresponding submit button is pressed, the database entry on the "EquipmentRulesAssignment" is updated with the new information of the operation and a new entry is added to the history table in order to create a record of all performed operations.

To access the history of performed operations to each equipment, a button was headed on the menu containing the equipment list. When this button is pressed, a new popup window 3.8 appears listing all the rules assigned to that equipment each one of them containing a table with all the operations performed to that equipment regarding that rule.

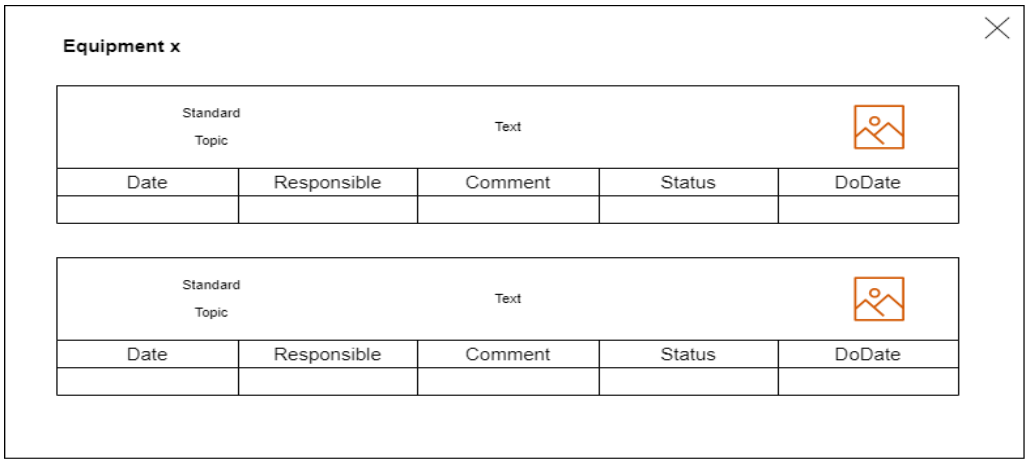
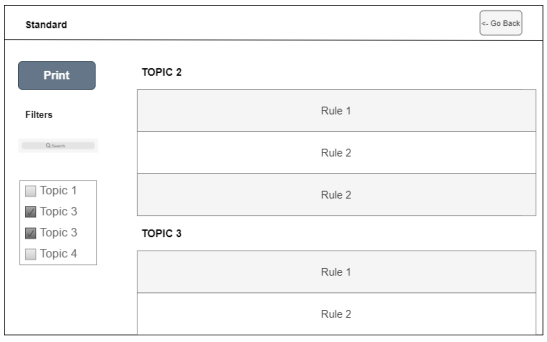


Figure 3.8: History mockup page. Implementation: 5.15

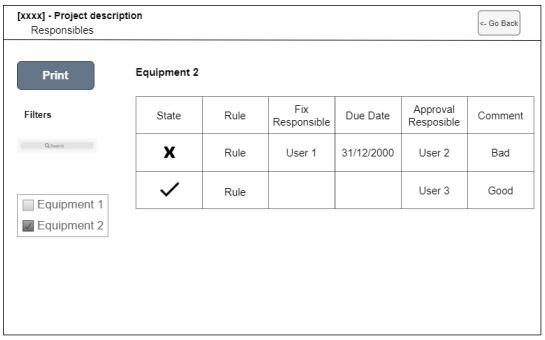
This feature is expected to be used and accessed by multiple users at the same time. So to prevent concurrency issues and loss of information, each time data from this tables of the database is loaded, added or updated, the user that requested it and the time frame when it happened is updated on the database. Using this method, if the requested information had been altered or requested by another user less than five minutes before the time that a new request is made, a warning will appear to inform that another person is working using that information and it will be locked from updates from the non already working user.

3.2.1 Printing

The possibility to print relevant information was one of the project requirements since the begging. This feature can be found on the page displaying the topics and rules of a given standard, allowing for the possibility to print it 3.9a. It can also be found on the page displaying the equipment present on a project, allowing for the creation of a report about the approved and rejected rules on the various equipment of a selected project 3.9b.



(a) Print standard mockup. Implementation 5.7



(b) Print equipment report mockup. Implementation 5.16

Figure 3.9: Print mode mockups

This was achieved using a custom JS function that when the print button is pressed, hides all the filter and unnecessary components, changes the font size for a more enjoyable user experience when reading and activates the browser printing functionality.

3.2.2 Authentication

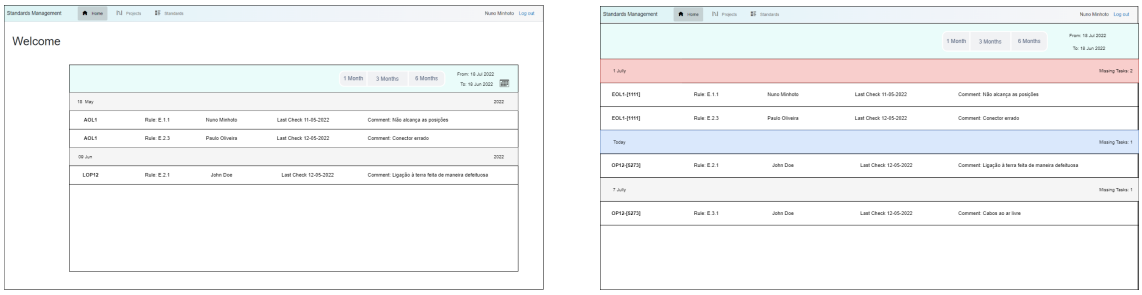
Regarding authentication and authorization, the approach chosen was to extend the already provided authentication state class from .Net complemented with JWTs for authentication on the DB and authorization on the server API.

To start, the user must insert the authentication credentials on the login page. If the credentials are valid, a request to server is sent to verify the identity of the user attempting to authenticate. The server API encrypts the password and attempts to find if that set of credentials correspond to a user that has access to this application. If that proves to be true, a new verification is done to verify the existence of that user on the corresponding table on the DB schema of this application. After this two verification's prove successful, a JWT is built with the corresponding user information. The claims that form the token consist on the 4 recommended ones (issuer, subject, audience and expiration time) and custom claims to identify the user name, control number and the permissions that are granted to the user's role. The expiration time granted to the token corresponds to thirty minutes, meaning that after that time the token is no longer valid. After the token is built, it is signed using an HMAC SHA256 algorithm finishing the token construction. After the token is complete, it is sent as a response for the login request to the client side, that saves the token on the browser storage and updates the authentication state to the user identity received on the token. To access the authenticated user information, the client side application checks the authentication state class for the claims corresponding to that user.

Since the token only has a lifespan of thirty minutes, a refresh method was implemented to keep the user session without the need for new manual authentication. When the server is adding the claims to the JWT, a random string is generated and added as a refresh token claim. This new token is also stored on the session table on the database, with a foreign key to the respective user and the expiration date for the refresh token, that is set for thirty days. With this, every time an expired JTW is used for any operation, the API will verify the corresponding session on the DB. If the token is still valid, and it is equal to the one stored on the JWT the session is refreshed, a new JWT is created and sent to the client with a new refresh token and expiration date, and the session DB entry is updated to the new token and a more time until expiration is added.

3.2.3 Dashboard

This feature was the last one to be developed, it was not present on the initial requirements for the project, but it was considered a nice to have addition during one of the meetings with the product owner. The goal is to display to the users the unapproved equipment rules from all the projects where they are one of the responsibles.



(a) Dashboard 1st mockup

(b) Dashboard final mockup

Figure 3.10: Dashboard mockups. Implementation 5.2

For this, the projects where the user is one of the responsibles are requested by the client. After that a list of the days that are due dates for rules approval is created and displayed 3.10b. The time interval to be displayed can be changed, but that does not affect due dates that are already late. Each task Contains information about the equipment in question and the project it belongs to, the rule that needs approval, the user that has to fix the problem in question, when the last visit was performed and the comment left on the last approval.

3.3 Encountered Problems

During the development phase, some problems rose on different stages. These were not major problems that prevented the continuation of development of the project, but they certainly took some time solving and getting through them.

The first one that was found was that the hot-reload feature provided by Visual Studio development environment was not working properly. The idea behind hot-reload consists on only reloading the files that where affected by a change introduced by the developer, instead of the whole application. The use of this method results on a more time efficient development since there is no need to recompile the whole code and all the changes can be seen quickly proving a good environment for UI design. Unfortunately, due to unknown reasons, it was not functioning correctly so every time we wanted to apply a change, the whole application had to be rebuilt and compiled.

The next obstacle appeared during the development of the interface. The accordion bootstrap component was not having the desired behavior when there where buttons present on it, making it a problem. No direct solution was found to that problem, but it was solved by having visually overlapped elements on the interface, fixing the behaviour problems that appeared initially.

Chapter 4

Testing

Testing is one of the most important steps in software development. In order to have a bug free application without any compromising errors a lot of tests have to be performed before releasing the application into the production environment.

This step can be performed using two different methodologies:

Automated Testing: Testing methodology that consists on the creation and use of scripts from automation frameworks to evaluate the expected output resulting from a set of predefined inputs. It is mostly used for massive testing on simple features where the result can be predicted and evaluated through scripts.

Manual Testing: Tests methodology performed by human interaction with the software. The testers have a list and description of features and tests for them to perform. It is usually used when automated testes cannot be created or when the evaluated functionality needs human evaluation and judgment.

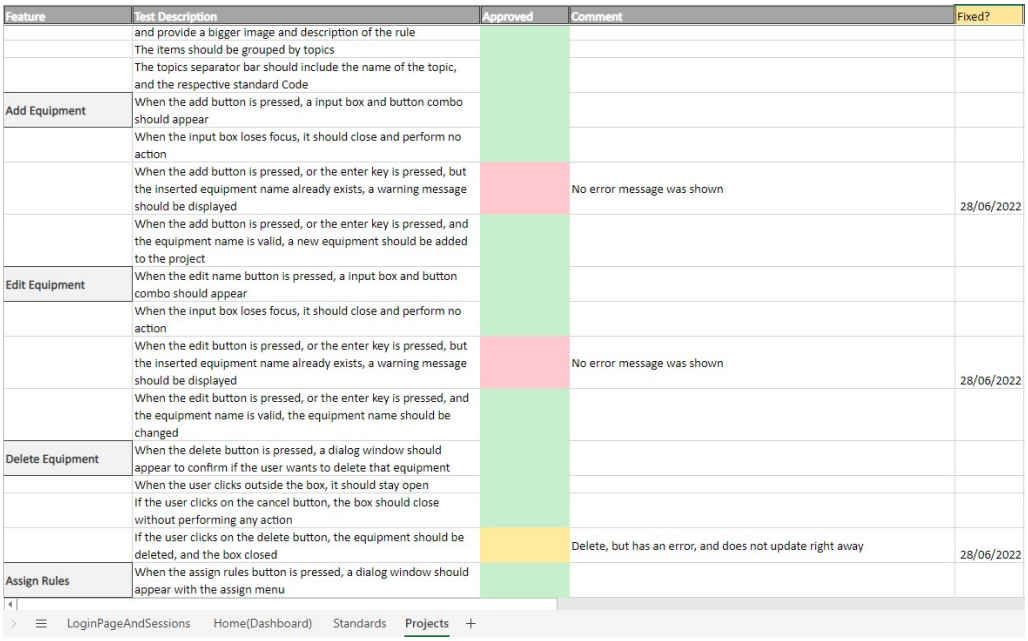
Through the whole development process testing was used to verify if the new features are working according to expectations and that older functionalities weren't affected or modified. Some parts of the web application, like server side features, could be tested through the use of automated testing, but most of it had to be tested manually since it involves a lot of human interaction. Due to time constrains, the full test plan was designed to be fully manual.

The Azure test plan tool was considered for the creation of the test plan. It provides a browser-based test management solution for exploratory, planned manual and user acceptance testing. But the use of this tool would present an added cost to the development since it is a payed resource of azure.

Taking this into consideration the test plan and management was performed using an excel spreadsheet.

4.1 Feature tests

The functionalities to test were defined in the excel document joined with dedicated columns to represent the result of each test and any comment that can be left with it.



Feature	Test Description	Approved	Comment	Fixed?
	and provide a bigger image and description of the rule			
	The items should be grouped by topics			
	The topics separator bar should include the name of the topic, and the respective standard Code			
Add Equipment	When the add button is pressed, a input box and button combo should appear			
	When the input box loses focus, it should close and perform no action			
	When the add button is pressed, or the enter key is pressed, but the inserted equipment name already exists, a warning message should be displayed		No error message was shown	28/06/2022
	When the add button is pressed, or the enter key is pressed, and the equipment name is valid, a new equipment should be added to the project			
Edit Equipment	When the edit name button is pressed, a input box and button combo should appear			
	When the input box loses focus, it should close and perform no action			
	When the edit button is pressed, or the enter key is pressed, but the inserted equipment name already exists, a warning message should be displayed		No error message was shown	28/06/2022
	When the edit button is pressed, or the enter key is pressed, and the equipment name is valid, the equipment name should be changed			
Delete Equipment	When the delete button is pressed, a dialog window should appear to confirm if the user wants to delete that equipment			
	When the user clicks outside the box, it should stay open			
	If the user clicks on the cancel button, the box should close without performing any action			
	If the user clicks on the delete button, the equipment should be deleted, and the box closed		Delete, but has an error, and does not update right away	28/06/2022
Assign Rules	When the assign rules button is pressed, a dialog window should appear with the assign menu			

LoginPageAndSessions Home(Dashboard) Standards Projects +

Figure 4.1: Excel testing document

Figure 4.1 shows some of the tests performed using this method. The feature and the description of the test to be performed are described on the left columns, and the result of each one on the right columns. The tests results can be represented into three categories:

- Green: Passed with the expected output.
- Yellow: Passed, but something went wrong.
- Red: Failed.

All the tests present in this document were performed and the adjustments to the software were done before the deployment of the application took place into the company servers.

4.1.1 Disadvantages

During the testing process the main difficulties and problems found were:

- Time consuming process, creating the test sheet and applying it correctly.
- It is hard to predict and specify all tests before using the application like a normal user will.

4.1.2 Advantages

Regarding advantages, these were the ones considered:

- Possibility to evaluate elements that need human judgement and opinion. Ex: User interface and experience.
- Better insight and description of the identified problem.
- New test items were thought and added while executing the plan.

Chapter 5

Results

After four months of development, the web application was in a usable and stable state to be deployed into the company .NET server.

5.1 Deployment

The deployment stage began with the recreation of the data model on the company database server that was then followed by the deployment of the web application into the .NET server, to be accessible and used by all the users. After everything was ready for use, the web application was presented to all the future users during a meeting.

5.1.1 Database

The creation of the database on the company's server was simple. All that was required was to run the migrations that were previously created for the local database during the development process. Some adjustments had to be made, since during development and exploration of the database a few tweaks were done manually on the database tables and data, but they were easily fixed by adding new migrations to include those adjustments.

5.1.2 Web Application

Since the application was developed using .NET6, and the servers were running version 5, an update to the server software had to be performed for it to be able to host it. After that, some problems occurred due to names given to the folders containing the application files on the server. This is was a common problem with the newer versions of .NET6, but it was easily fixed by adapting the folder names and thus creating a safer file path.

After having the application up and running, while performing the test plan to make sure that every feature was performing according to expectations, some other problems were encountered. When trying to upload files through the application an error would appear due to the it not being able to find the image on the given path. This error was resolved by manually creating the folders for those images to be stored, since they were not created previously at the time of the deployment.

Another occurring error appeared when trying to update or delete information on the database. This was a permission configuration error in one of the files and was quickly fixed.

After all the tests on the excel sheet were passed by the deployed application, it was granted access to the company employees to explore and experiment the application as well as start populating the DB with data.

The only page accessible of the web application to a non authenticated user is the login page, figure 5.1.

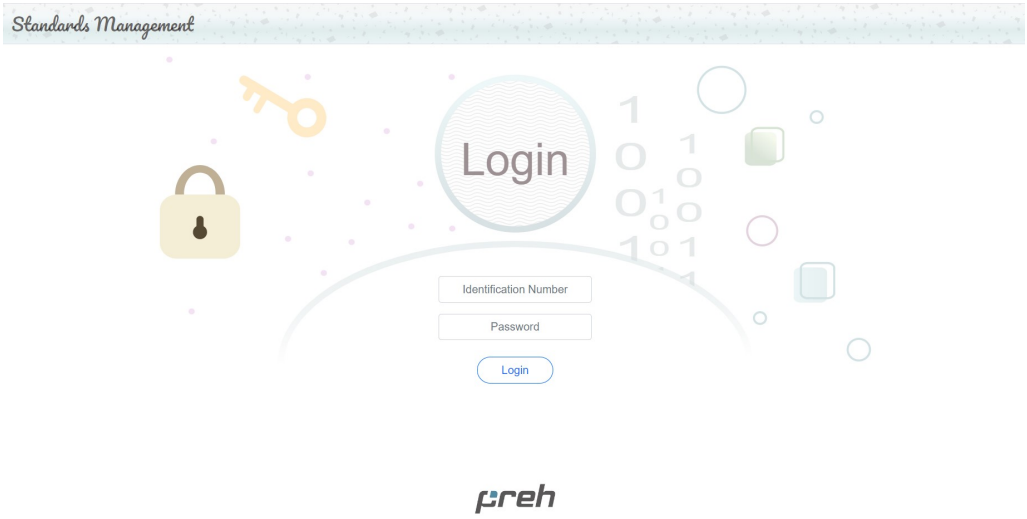
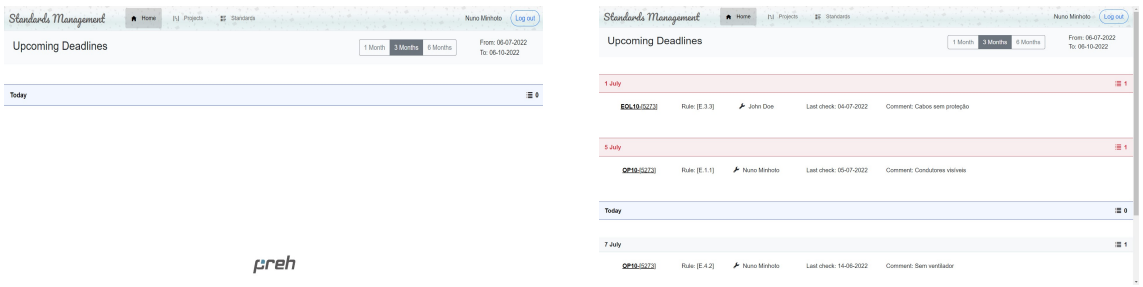


Figure 5.1: Login Page

In here, the only available action for the users to perform, is to authenticate on the web application.

After the authentication is performed, the users will have access to other pages and functionalities. Once this process is complete, the users are taken to the home page, where the dashboard is presented, figure 5.2, and since the users are already authenticated, they now have access to the navigation bar functionalities, granting them access to other pages, and the ability to perform log out.

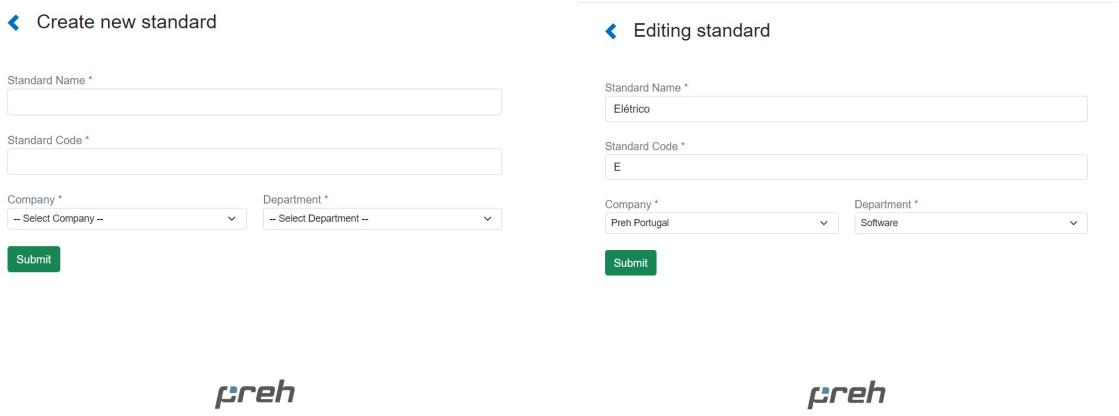


(a) Empty dashboard (b) Example dashboard

Figure 5.2: Home page display

When the user first logs into the web application, the dashboard will be empty, figure 5.2a, since he is not responsible for any projects. But after being added as a responsible to any project, the due dates of equipment present on that project will start to appear on the board, figure 5.2b.

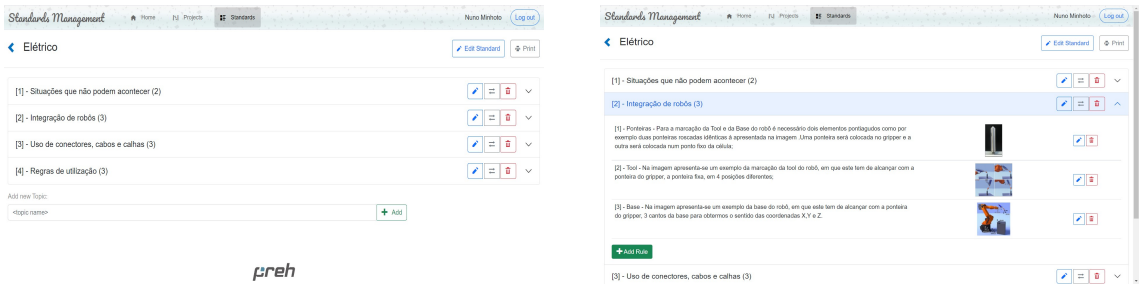
Through the navigation bar the list standards page can be accessed. On this page, a list of all standards is presented. If the role granted to that user has administrator permissions, he will be allowed to create, edit and delete standards.



(a) Create standard example (b) Edit standard example

Figure 5.3: Create and Edit standard interface

Through this page, it is also available for the users to access more information about the required standards, figure 5.4.



(a) Standard Display (b) Standard Rules display

Figure 5.4: Standard information display

As it can be seen on figure 5.4b the rule information is stored and grouped by topic on the standard display page. This will be the main method for users to consult and view the rules with ease, having the possibility for users to have a better view of a specific rule by clicking on the corresponding image, figure 5.5.

If the user has administrator permissions, he will see have access to the create, edit and delete topics and rules functionalities, allowing them to manage the information present on the database, figure 5.6.

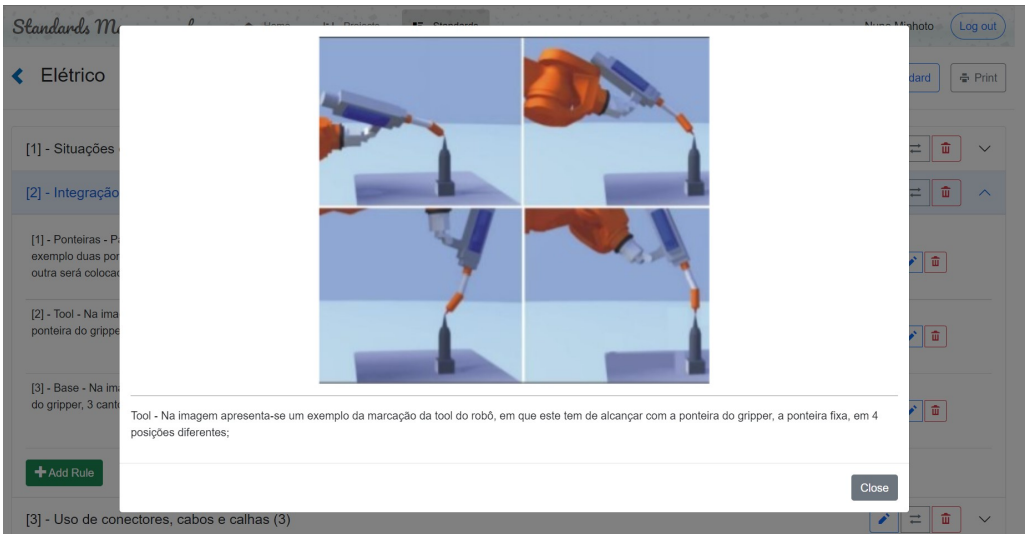
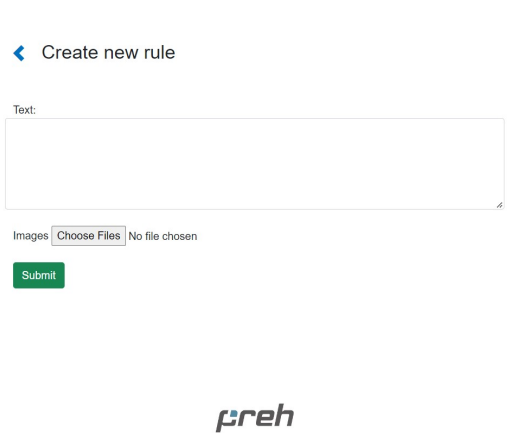
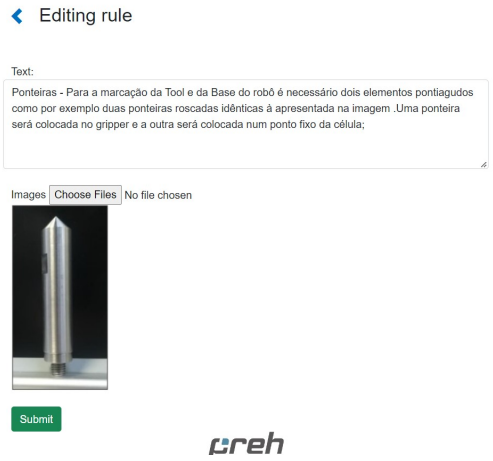


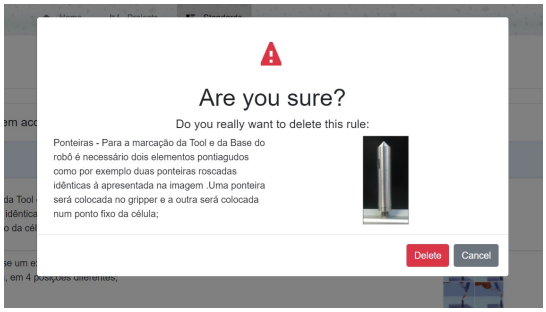
Figure 5.5: Detailed rule



(a) Creating rule example



(b) Edit rule example



(c) Delete rule

(d) Edit topic example

Figure 5.6: Managing standard information examples

The last possible functionality that is accessible through this page, is the print standard, 5.7.

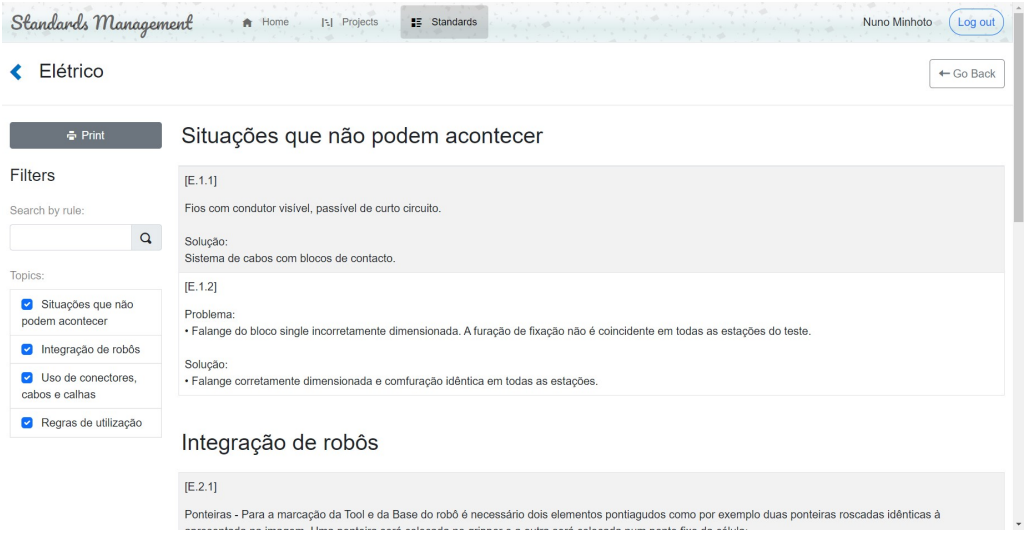


Figure 5.7: Print Standard example

Through this, the users are allowed to create a pdf file containing the rules selected by them. This page has a dedicated filter column where it is possible to select which rules, topics and standards are going to be printed.

The last feature can only be accessed through the navigation bar, on the projects button. After pressing that button the user is redirected to a page where the projects under his responsibility are listed, figure 5.8, or all projects are listed, if the user has a specific permission.

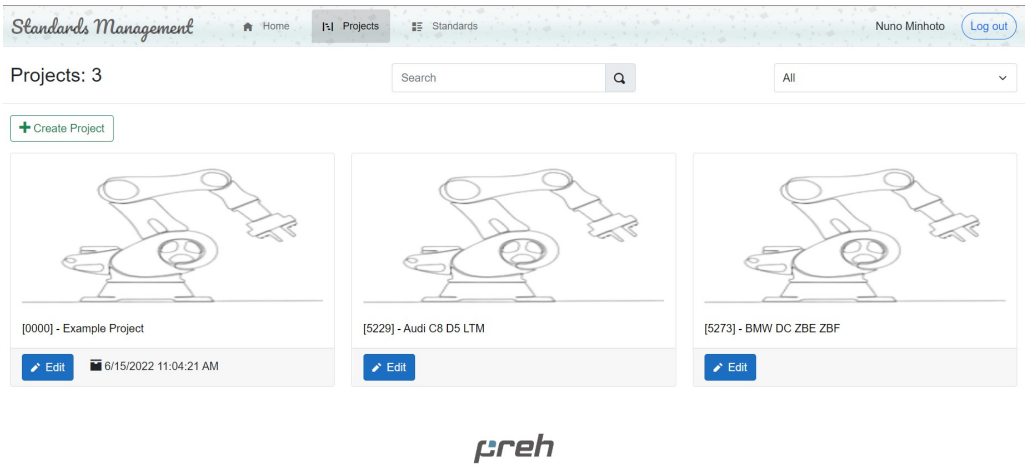


Figure 5.8: Project display example

On this page, the users with certain permissions can create, edit and archive projects on a similar model as it was done for the standards and rules. If no image is assigned to a project upon creation, a default image is displayed on the page. Pressing on a project card will redirect to a new page displaying the information relative to a single project.

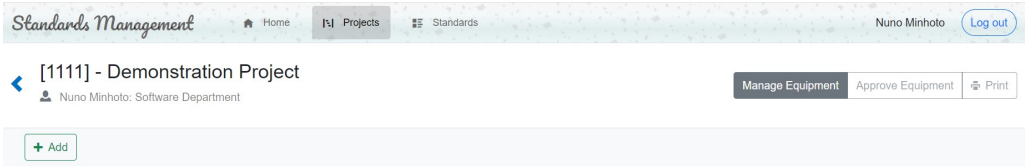
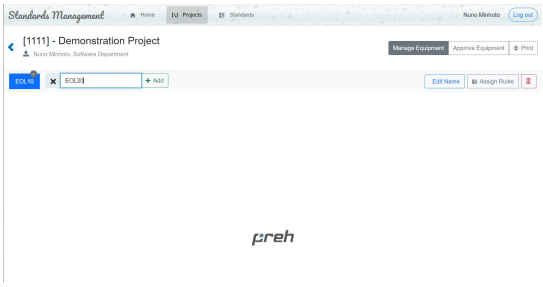
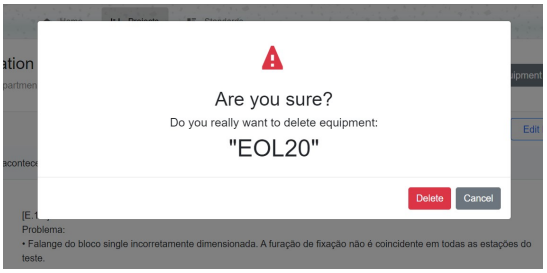


Figure 5.9: Blank project example

When the project is created it starts of empty and without any equipment, so it is possible to add equipment on the management mode on this page.



(a) Adding equipment example



(b) Delete equipment example

Figure 5.10: Managing equipment example

After the equipment has been created, the option to assign rules is available. It is thought this method, displayed on figure 5.11, that the link between equipment and rules is made.

On figure 5.12 it is shown an example of a project with equipment that have some rules assigned already. Now it is possible to start performing the approval operations by going to the approval mode, figure 5.13.

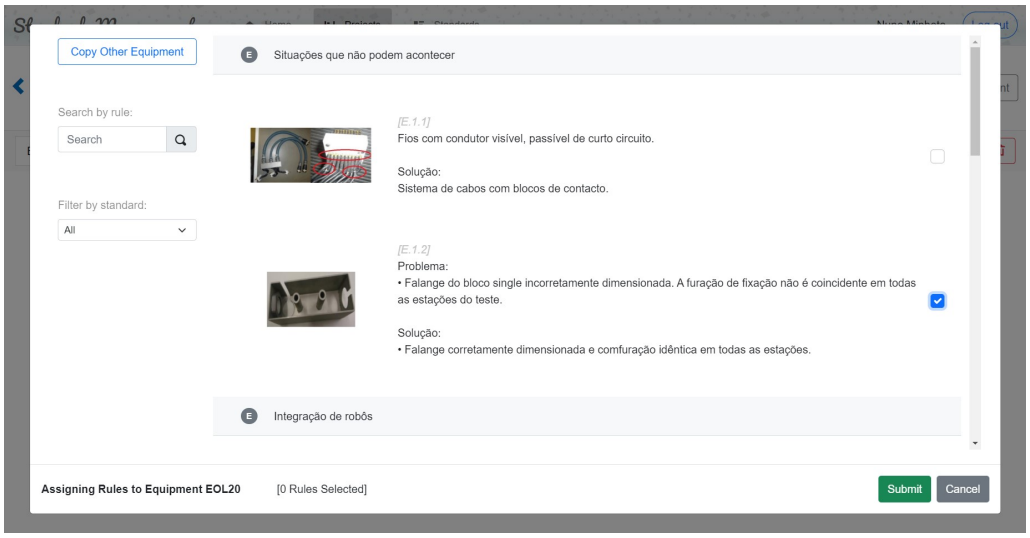


Figure 5.11: Assign rules to equipment example

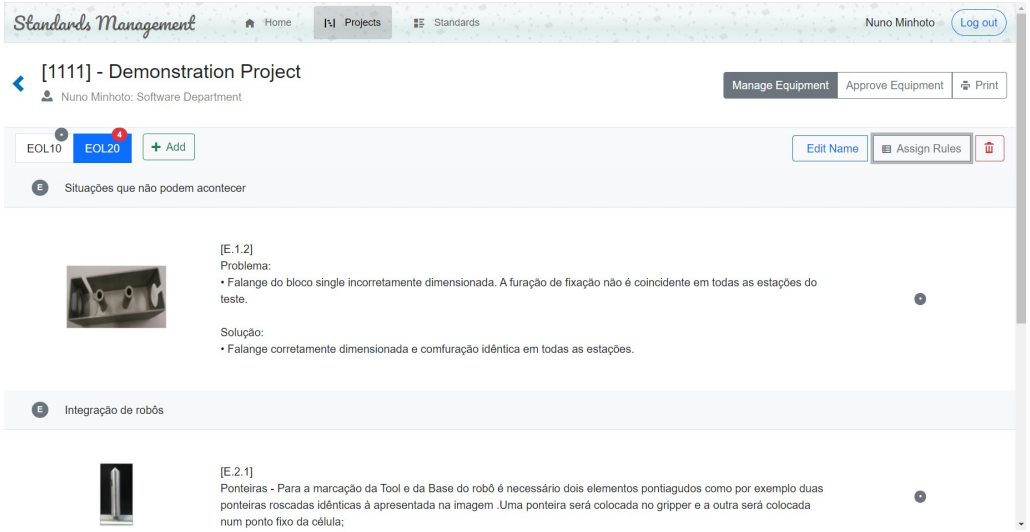


Figure 5.12: Project equipment display

It is here that the users now have the possibility to either approve or reject, the rule depending on the verification they perform on the physical equipment.

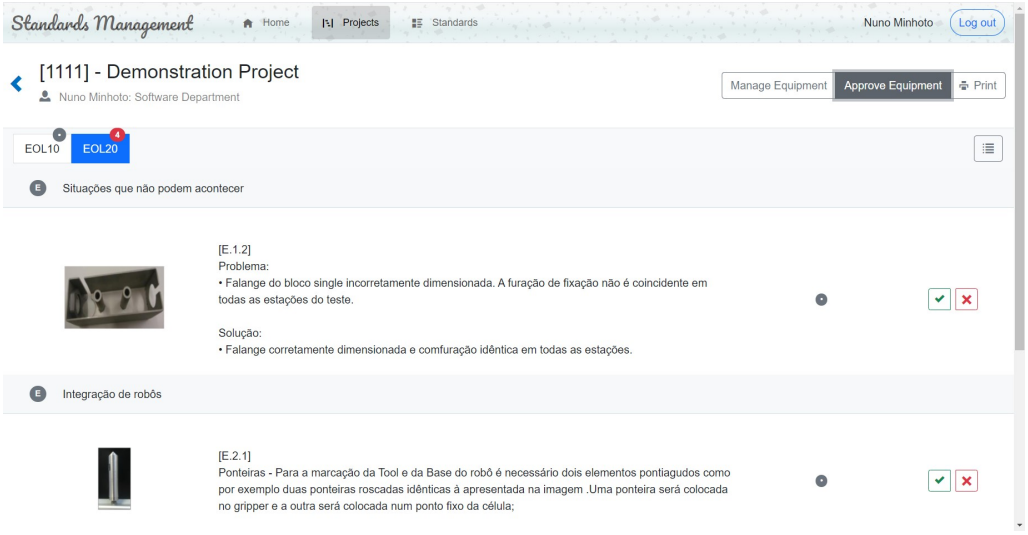
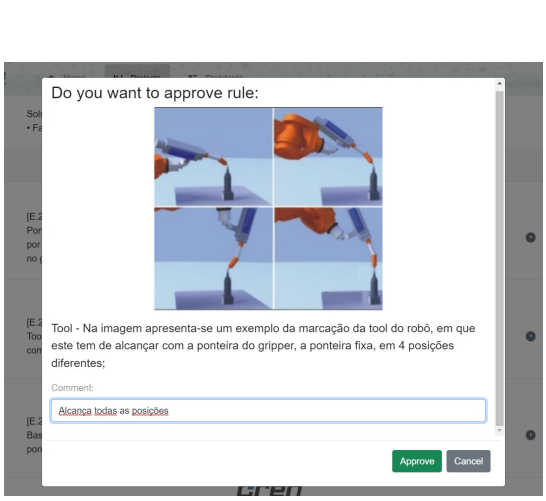
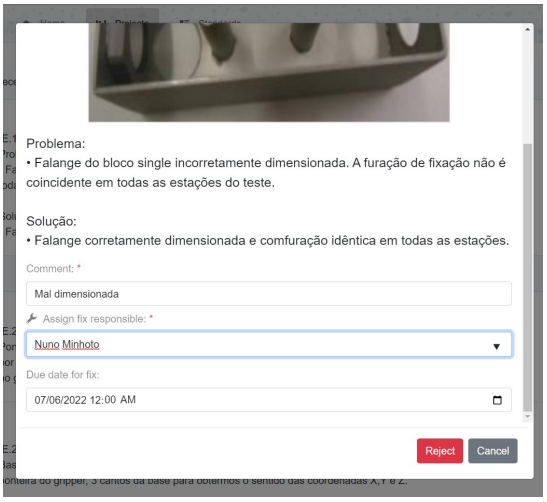


Figure 5.13: Approve mode example



(a) Approve rule example



(b) Reject rule example

Figure 5.14: Approval operations example

This mode also features a history consultation mode, allowing for the last performed operations on a specific equipment to be viewed.

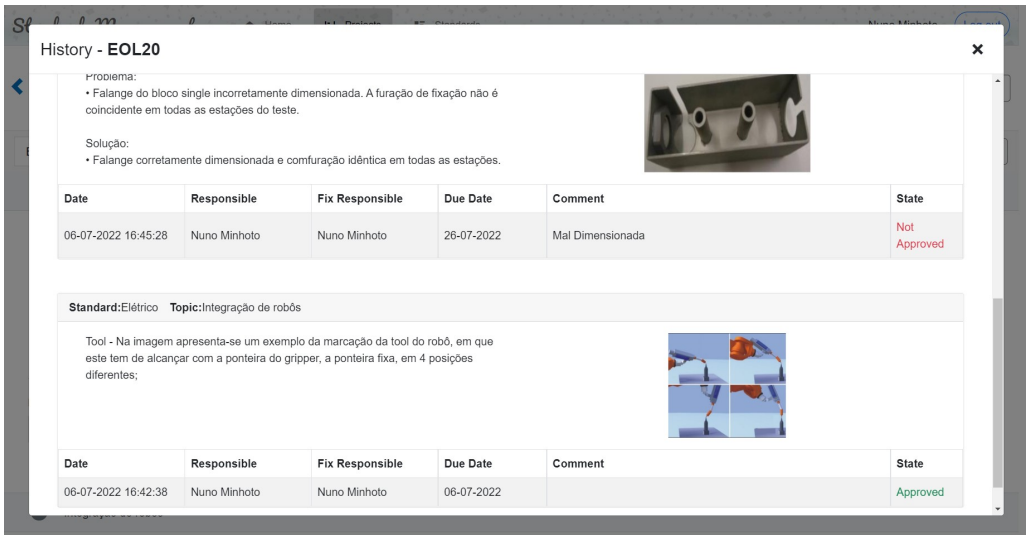


Figure 5.15: Equipment history example

The last available mode on this page, is the print mode. Just like the print option for the standards page, this has a dedicated column for the filtering options. It allows to filter by equipment, responsible person for fixing a rule, specific rules and by standard.

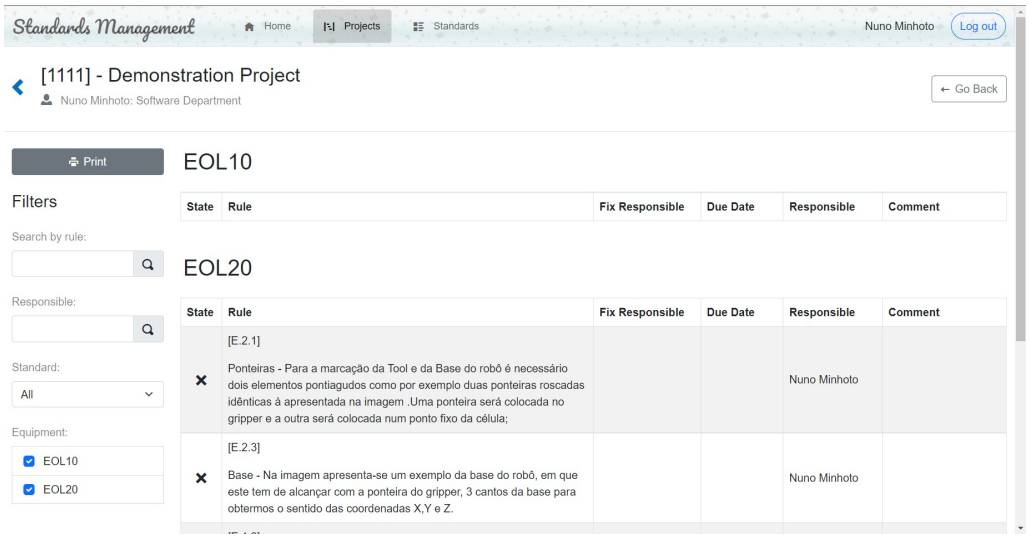


Figure 5.16: Equipment print example

5.2 User testing

When the application was deployed, some of the company workers were granted access to perform some tests and provide feedback. The first step that was taken was for one of the users, André Marques, to populate the database with the information contained within the electrical standard. This translates on analysing the documents containing the standards information and using the app dedicated tab to upload that information into the database. This process occurred without any problems.

Even if the target users of the application are the employees from Preh Portugal, the application is going to be used while they are out doing approval verification's on other devices that are present on different facilities. Since the application is hosted on the company server, it can only be accesses from within the intranet, therefore to access the application from outside the network a virtual private network needs to be used. With a VPN the employees can be connected to the network and access the web app normally.

On the first test on a real situation for the application on a different facility, due to bad internet connection the performance of the application was affected. Most of the performed testing was executed while connected directly to the company's network therefore this was a problem that was not faced before.

5.3 Users Feedback

User feedback and information is a big aspect during the development, since they are the ones that are going to utilize it the application the most. After the web application was presented and open for the users to use and experiment, the received feedback was used to create the following statistics and graphs.

This statistic was elaborated from the 3 anonymous replies from the universe of 4 users that were granted access. The feedback questions contains qualitative global questions that were answered by choosing a number between 0 (very bad/least likely) and 5 (very good/most likely), on table 5.1 the average of the replies is presented.

Table 5.1: Feedback table

Global appreciation of the application	4.33
Do you think the application is user friendly and easy to use?	4
How would you rate the features present on the application?	4
How would you rate the application design globally?	4
Rather use it in substitute of the old method?	5

To evaluate the appreciation of the features and the developed pages the graphics on figures 5.17 and 5.18 were built with the obtained replies.

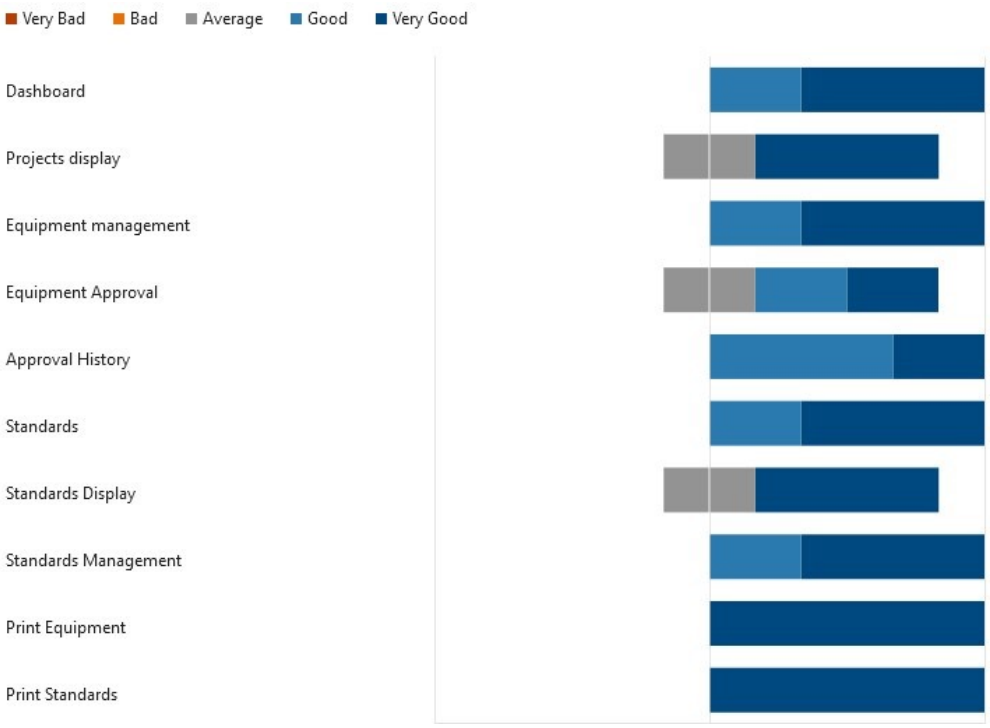


Figure 5.17: Feedback graph on the developed features

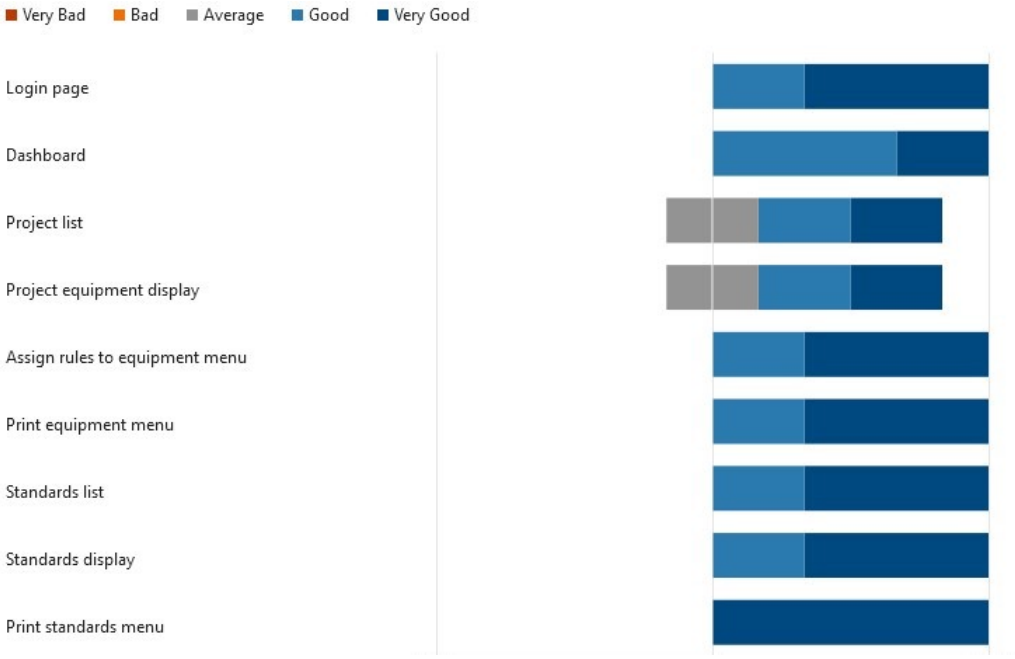


Figure 5.18: Feedback graph on the design

Lastly there are open answer questions where the users can give their opinion on the inquired subject.

Question: Do you think the application will reflect an improvement regarding the previous approval method? Why?

- The previous method was hard to search the rules for each equipment and with this project we will have only the necessary rules associated with each device.
- Yes. The approval process is now faster and more "accurate".
- The application reflects an improvement on the system previously used as it allows centralised management of the different standards used by Preh and facilitates the equipment validation procedure.

Question: What was your favorite feature/aspect of the application?

- The scheduler at the first page make it easy to know when something needs to be approved.

Question: Any suggestions on how to improve the current features?

- At a remote site the application should perform better, in-house the performance is good for now.
- Add an option to filter by topic when the equipment rule list is displayed.

Question: Any new features you would like to see included?

- Merge with current systems that create the projects for quotations.
- Tag system for the standard information.
- Include images when printing standard information.
- Improved sorting and filtering options in order to make the search for information easier.
- Rating by standard item. I would like to see which items have the highest rejection rate and why.

Question: Any disliked feature?

- No.
- No, all the existing functionalities are useful and essential for a better management of standards.

Question: Any suggestions on how to improve the current interface?

- Remove some of the clutter and maybe the approve equipment action should not need a confirmation window.

Question: Do you think the use of this application will save time?

- This application will save a lot of time for managing the standards and to approve the equipment on our suppliers.
- Yes.

Chapter 6

Conclusion and future work

The following chapters consist of a synthesis of all the steps taken during the development of the project associated with this dissertation and the future work that was considered for it.

6.1 Conclusion

The project associated with this dissertation was developed for the company Preh Portugal that created production equipment for the automotive industry that have to comply with several rules. The method used previously to ensure that this rules were followed consisted on consulting the power point files containing the standard rules and creating a shortened version of the rules on an excel document to be approved, something similar to the testing method used for this application. This lead to the idea for the creation of a new platform that could facilitate this process of managing and verifying if the standard rules were being followed, offering a more enjoyable experience to the workers that will perform those operations.

Taking the feedback from the users of the previously method, a requirement gathering was performed for a web application that allows for a control and enforcement of the current standards to a given equipment being designed or updated, which can be seen on section 1.3.

With those requirements in mind the necessary set of tools and decisions on which ones to use were made ending up on the following being chosen:

- **Platform:** Web Application
- **Framework:** Blazor WebAssembly from ASP.NET present in the .NET framework, where version 6 was used.
- **Data storage** Database hosted with Microsoft SQL Server, using Entity Framework for Object-relational Mapping.

After having made all the decisions regarding what technologies where going to be used, the development work was performed during the following months. After the development process,

the web application was tested in order to make sure it complied with the requirements. This resulted on the final application that was deployed and made available for all the workers to use.

From the initially gathered requirements, the following were validated:

1. **Should use the already existing user data base for authentication:** The web application currently uses the already existing company user database for authentication and authorization purposes.

2. **Should have a role and permission system, where each user has only one role:** Each user has a specific role assigned. Each role have some permissions assigned to it, and it is those permissions that can have an effect on which functionalities or pages are accessible to the each user role. The currently existing roles are:

   - Developer: Access to all information and info that is available on the application.
   - Admin: Has editing permissions for standards and projects.
   - User: Can view the standard information and perform approval operations on equipment of projects he is responsible of.

3. **Multiple users should be allowed to work simultaneously, but if they try to work on the same project, they should be notified:** The web application does this by verifying when the data from the database was last loaded, and if time interval is small, displays warning information.

4. **Support for creation and editing of new standards:** There are dedicated pages on the web application for adding, editing, viewing and deleting standards.

5. **Should allow to insert or edit topics and rules for each standard:** On the page to view the information about on standard, there are present options for the creation of topics and rules.

6. **Each rule should be allowed to contain text information and images:** Rules are stored on the database by saving the corresponding text and the list of images associated with them.

7. **Should allow for standards and the corresponding topics to be printed:** A print option is present on the view standard page, allowing it to be printed.

8. **Should allow for projects to be created, edited and viewed by the users using User Access Rights:** One of the main pages of the web application lists the projects where the logged in user is one of the responsibles, and grants them access to the page displaying all the information related to that project. This page also allows to create and edit projects.

9. **The users should be able to create equipment on the respective projects:** On the page displaying the information about one project, users can create the equipment for that same project.

10. **After the equipment is created, rules should be able to be assigned to them:** Every created equipment has an option to add or remove assigned rules to it.

11. **Should have an approval mode where the users can indicate if the rules are being followed by the physical equipment or not:** There is a dedicated approval mode on the project information display page, where the main functionality is to verify if that rule is approved or not.

12. **On the approval mode, a set of information about the assignment of a rule to an equipment must be displayed:** When a rule assignment with an equipment is displayed, it displays relevant information regarding that assignment.

All the objectives defined initially were met, and the final result of the project developed during this dissertation consisted on a web application that allowed tracking and documenting hardware and software operations that are performed on production equipment. It allows the users to save time on the preparation and during the approval operation all while maintaining a simpler a easier documentation of all the standards and performed operations.

6.2 Future Work

When it comes to future work, the expected is to continue to provide support while the transition from the old method is being executed to this new platform fully. Also, having the feedback received as a guidance, improving and developing further the already exiting features and add other ones that may appear as suggestions like the following:

- Add integration with other platforms currently being used at the company.
- Improve the performance when the application is being used on a remote location.
- Improve the user interface.
- Review the method to confirm an approved rule.
- Design and implement a tag system for the standards information.
- Add the possibility to print images to complement the standard information.
- Add more sorting and filtering option to the menus.

In order to make this process of adding or changing new features quicker, the implementation of a CICD pipeline is considered, since the current method for testing and deploying the changes added is fully manual.

CI/CD

In software engineering, CICD is the combination of practices of Continuous Integration (CI) and Continuous Delivery or Continuous Deployment (CD). This bridges the gap between development and operational activities by enforcing automation in testing and the deployment of the web application. This has the aim to increase error detection, increase productivity and provide faster release cycles. The adoption of a mechanism like this would make the process of correcting or adding features faster and easier, since the current method is performed fully manually.

References

[1] Dragos-Paul Pop and Adam Altar. Designing an MVC model for rapid web application development. *Procedia Engineering*, 69:1172–1179, 2014.

[2] Peter Himschoot. *Microsoft Blazor*. Building Web Applications in .NET. Apress, 2020.

[3] Shashank Mohan Jain. *WebAssembly for Cloud*. A Basic Guide for Wasm-Based Cloud Apps. Apress, 2022.

[4] Sebastian Peyrott. The JWT handbook. *Auth0 Inc.: Bellevue, WA, USA*, 2017, 2016.

[5] Daniela Berardi, Diego Calvanese, and Giuseppe De Giacomo. Reasoning on UML class diagrams. *Artificial intelligence*, 168(1-2):70–118, 2005.

[6] Julia Lerman. *Programming Entity Framework: Building Data Centric Apps with the ADO. NET Entity Framework*. " O'Reilly Media, Inc.", 2010.

[7] Jeffrey Richter. *Applied Microsoft. NET framework programming*, volume 1. Microsoft Press Redmond, 2002.

[8] Juyun Joey Cho. An exploratory study on issues and challenges of agile software development with scrum. *All Graduate theses and dissertations*, page 599, 2010.

[9] Heonsik Joo. A study on understanding of UI and UX, and understanding of design according to user interface change. *International Journal of Applied Engineering Research*, 12(20):9931–9935, 2017.

[10] Jean Vanderdonckt and Costin Pribeanu. State of the art of web usability guidelines. 2005.

[11] Jennifer Niederst Robbins. *Learning Web Design: A Beginner's Guide to (X) HTML, StyleSheets, and Web Graphics*. "O'Reilly Media, Inc", 2007.

[12] David Flanagan. *JavaScript: the definitive guide*, volume 1018. O'reilly, 2006.

[13] Anders Hejlsberg, Mads Torgersen, Scott Wiltamuth, and Peter Golde. *The C# programming language*. Pearson Education, 2008.

[14] Jake Spurlock. *Bootstrap: responsive web development*. "O'Reilly Media, Inc", 2013.

[15] F. Ricca and P. Tonella. Analysis and testing of web applications. In *Proceedings of the 23rd International Conference on Software Engineering. ICSE 2001*, pages 25–34, 2001.

[16] Roy Miller and Christopher T Collins. Acceptance testing. *Proc. Xp Universe*, 238, 2001.

[17] Alexander Poth, Mark Werner, and Xinyan Lei. How to deliver faster with CI/CD integrated testing services? In *European Conference on Software Process Improvement*, pages 401–409. Springer, 2018.

[18] Paul Pop. Comparing web applications with desktop applications: An empirical study. *Linköping University, Linköping*, 2002.

[19] Aaron Marcus. Principles of effective visual communication for graphical user interface design. In *Readings in human–computer interaction*, pages 425–441. Elsevier, 1995.

[20] Juha Itkonen, Mika V. Mantyla, and Casper Lassenius. How do testers do it? an exploratory study on manual testing practices. In *2009 3rd International Symposium on Empirical Software Engineering and Measurement*, pages 494–497, 2009.

[21] San Murugesan. Web application development: Challenges and the role of web engineering. In *Web engineering: modelling and implementing web applications*, pages 7–32. Springer, 2008.

[22] K. Gutzmann. Access control and session management in the http environment. *IEEE Internet Computing*, 5(1):26–35, 2001.

[23] Corrado Aaron Vlsaggio and Lorenzo Convertito Blasio. Session management vulnerabilities in today's web. *IEEE Security Privacy*, 8(5):48–56, 2010.