**UNIVERSIDADE FEDERAL DE SANTA CATARINA**
**CENTRO TECNOLÓGICO**
**DEPARTAMENTO DE AUTOMAÇÃO E SISTEMAS**

**Jean Caetano Perufo Damke**

# Modeling and Development of a Web Application for Data Analysis of Air Systems on Jet Engines

Florianópolis
2018

**Jean Caetano Perufo Damke**

# Modeling and Development of a Web Application for Data Analysis of Air Systems on Jet Engines

Relatório submetido à Universidade Federal de Santa Catarina como requisito para a aprovação na disciplina **DAS 5511: Projeto de Fim de Curso** do curso de Graduação em Engenharia de Controle e Automação.
Orientador: Prof. Ricardo José Rabelo

Florianópolis
2018

**Jean Caetano Perufo Damke**

# Modeling and Development of a Web Application for Data Analysis of Air Systems on Jet Engines

Esta monografia foi julgada no contexto da disciplina DAS5511: Projeto de Fim de Curso e aprovada na sua forma final pelo Curso de Engenharia de Controle e Automação.

Florianópolis, 30 de Julho de 2018

**Banca Examinadora:**

Ricardo Grützmacher
Orientador na Empresa
Rolls-Royce Deutschland LTD CO & KG

Prof. Ricardo José Rabelo
Orientador no Curso
Universidade Federal de Santa Catarina

Prof. Leandro Buss Becker
Avaliador
Universidade Federal de Santa Catarina

Matheus Alberto Ambrosi
Debatedor
Universidade Federal de Santa Catarina

Gabriel Monteiro de Souza
Debatedor
Universidade Federal de Santa Catarina

## RESUMO

O trabalho apresentado neste documento foi realizado na empresa Rolls-Royce Deutschland CO & KG, na sede localizada em Dahlewitz, na Alemanha. A Rolls-Royce Deutschland é uma subsidiária da Rolls-Royce Plc, uma fabricante de sistemas de potência para diversos setores da indústria. A Rolls-Royce Deutschland (RRD) é um dos ramos focados em aviação, desenvolvendo motores a jato de diversos portes, e possui duas sedes, uma em Dahlewitz, na região metropolitana de Berlin, e uma em Oberusel, próximo de Frankfurt am Main.

O presente trabalho foi desenvolvido no departamento de Secondary Air Systems, responsável pela modelagem dos fluxos de ar secundários dos motores a jato, que são utilizados por vários outros sistemas.

O problema atacado por este trabalho foi a dificuldade de visualização e comparação de dados dos sistemas de ar secundários. Atualmente a visualização e comparação é feita através de planilhas Excel, que limitam o trabalho em vários quesitos.

A solução apresentada foi a modelagem e desenvolvimento de um sistema web para comparação e visualização destes dados. Este sistema tomou como base a aplicação desenvolvida pelo aluno e por outro estagiário ao longo do semestre que precedeu o semestre de desenvolvimento deste PFC, e que continuou sendo aprimorada ao longo deste. Ambas as aplições são focadas na apresentação de dados através de ilustrações técnicas interativas e gráficos, e ambas foram desenvolvidas utilizando as mesmas tecnologias, das quais as principais são as linguagens Elixir e GraphQL e a biblioteca ReactJS.

A metodologia do trabalho foi um misto de desenvolvimento ágil com ferramentas clássicas de modelagem UML. Tal abordagem se deu pela impossibilidade de se realizar uma metodologia ágil pura, como foi a aplicação do semestre anterior. Isso ocorreu devido à ausência da pessoa do departamento de Secondary Air Systems que estava supervisionando o projeto. Por tal motivo, o trabalho passou por um período de 2 meses em que não houve possibilidade de realizar reuniões ou consultas rápidas, o que seria fundamental para a aplição de uma metodologia ágil ao longo do projeto todo.

Como resultado, o trabalho gerou a modelagem completa de uma aplicação que resolve o problema descrito nos parágrafos anteriores, além da implementação de uma versão prévia dessa aplicação. Segundo estimativas, tal aplicação possui o potencial para gerar uma grande redução de custo para a empresa, além de aproximá-la de uma integração entre os diversos sistemas corporativos dela.

**Palavras-chave**: Rolls-Royce, Aplicação Web, Elixir, React, GraphQL.

## ABSTRACT

The work presented in this document was developed in Rolls-Royce Deutschland CO & KG, Dahlewitz site, Germany, a manufacturer of jet engines. The work was made inside the department of Secondary Air Systems, which is responsible for the modelling of the secondary air flows inside these types of engines. The problem that this work aimed to solve was the visualization and comparison of data. Currently, this task is made via Excel spreadsheets, which is a very limited tool. The solution presented was the modelling and development of a web application which took as base the application developed in the previous semester by this student and another intern. Both applications are focused on presenting data through technical illustrations and graphs, and both were developed using the same technologies, whose main ones are the language Elixir and GraphQL and the library React. The methodology implemented was a mix of Agile and standard UML practices, a combination that brought as a result a complete model of the application in a short period of time, and an initial version of it already implemented.



**Key-words**: Rolls-Royce, Web-application, Elixir, React, GraphQL.

## LIST OF FIGURES

Figure 1: Schedule March/April ..... 22
Figure 2: Schedule May/June ..... 23
Figure 3: Schedule June/July ..... 23
Figure 4: Use-Case Diagram ..... 27
Figure 5: Layout Navigation Bar ..... 29
Figure 6: Layout TDM Screen ..... 30
Figure 7: Layout TDM Measurement Units ..... 30
Figure 8: Layout TDM Validation ..... 31
Figure 9: Layout Validation Graph ..... 32
Figure 10: Layout Plot Screen – Parameter Selection ..... 32
Figure 11: Layout Plot Screen ..... 33
Figure 12: Layout SPAN Screen ..... 34
Figure 13: Conceptual Entity-Relationship Model ..... 37
Figure 14: Database ..... 38
Figure 15: Mutation ..... 39
Figure 16: Schema Type ..... 40
Figure 17: Simple Query ..... 40
Figure 18: Complex Query ..... 41
Figure 19: Query Resolver ..... 41
Figure 20: SPAN Screen ..... 42
Figure 21: SPAN Screen with Graph ..... 43
Figure 22: SPAN Screen with Sidebar and Bottombar collapsed ..... 43
Figure 23: TDM Screen ..... 44
Figure 24: Flow Values Query ..... 46
Figure 25: Test Files Query ..... 47
Figure 26: TDM Boxes Query ..... 48

## LIST OF ACRONYMS

BU – Basic User

ER – Entity Relationship

ERF - Experiment Request Form

JS - Javascript

PFC – Projeto de Fim de Curso

RR – Rolls-Royce

RRD - Rolls-Royce Deutschland

SAS – Secondary Air Systems

SLN – Solution (Document)

SPAN – Suite of Programs for Air Networks

SPU – Special User

SU – Super User

TDM – Test Data Manager

UFSC - Universidade Federal de Santa Catarina

UML – Unified Modeling Language

URL – Uniform Resource Locator

**SUMMARY**

## Contents


1 Introduction ..... 10
1.1 Main Goal ..... 10
1.2 Specific Goals ..... 10
1.3 Technology Stack ..... 11
1.4 Methodology ..... 11
1.5 Solution and Alternatives ..... 11
1.6 Graduation ..... 12
1.7 Next Chapters ..... 12
2 Company, problem and Solution ..... 13
2.1 The Company ..... 13
2.2 The Problem ..... 14
2.3 The Solution ..... 14
2.3.1 Methodology ..... 15
2.3.2 Technology Stack ..... 15
2.4 Concepts ..... 18
2.4.1 Initial Concept ..... 18
2.4.2 The Future ..... 18
2.5 Methodology ..... 18
3 Modelling ..... 20
3.1 Tasks ..... 20
3.2 Schedule ..... 22
3.3 System General View ..... 24
3.4 Use Case Diagram and Requirements ..... 26
3.5 Layout Definition ..... 29

3.6 Database ..... 34
4 Implementation ..... 39
5 Results ..... 42
5.1 Verification ..... 44
5.1.1 Database check ..... 44
5.1.2 Functionality and Requirements Check ..... 48
6 Considerations and Perspectives ..... 51
6.1 Result Analysis ..... 51
6.2 Future Perspectives ..... 52
References ..... 53
Appendix A – Table of Requirements ..... 55

# 1 INTRODUCTION

The work presented in this document aims to solve a problem inside the department of Secondary Air Systems of Rolls-Royce Deutschland CO & KG. The issue is the visualization and comparison of data related to tests and simulation of air flows on jet engines, which is currently done using Excel spreadsheets, a very limited tool.

The selection of the subject was made from two different perspectives. One from the company, that needed a relevant project that solved or improved one process of the company. Also, it would be interesting if the student could keep working in something related to the software developed in the semester before, and this software would be somewhat of an extension of the previous one. From the student's side, a project that encompassed many areas of study inside the Control and Automation Engineering course was desired, because this would mean that the student would have the oportunity to gain experience in many areas.

## 1.1 Main Goal

The main goal of the work developed was to improve one task inside the company, which, as already mentioned, is the visualization and comparison of test data with simulation data in air systems of jet engines. This task is part of two different processes related to the validation of models and certification of engines. This means that these processes are crucial for the company, and improving it is of utmost importance for the goal of increasing the efficiency of the company as a whole.

The goal of improving this task was based in the current low-efficiency of the process, which is done via excel spreadsheets, and thus comes with a lot of limitations such as being available for a single project, the impossibility of showing only some data to specific users and of integrating with other systems inside the company.

## 1.2 Specific Goals

Diving into the more specific goals, the objective was to develop a web application, which was named Engino TDM, that enabled the employees to achieve the main goal, described in the section above, while being easy to use, easy to maintain and upgrade, and easy to integrate with other systems.

## 1.3 Technology Stack

Since this application would reuse some of the code from the application developed in the previous semester, called Engino, it was decided that the technology stack and other aspects involving the modeling of the application presented in this thesis would be similar to the ones chosen for Engino. This was necessary for its maintainability, since the future employees who would be responsible for Engino would also be responsible for this application, and thus it is required that both application share the same tech stack in order to prevent an overhead of language requirements for those employees.

## 1.4 Methodology

The methodology used in this project was a mix of agile development with practices from UML. This approach was selected because it was not possible to have regular encounters with people who understood the problem during the whole period of the work, something that would be crucial if a pure agile methodology was selected. On the other hand, the project is a web application and does not required a large amount of documentation, since it was developed by a single person with the counseling of another, and it needed to be delivered as soon as possible. Because of these reasons, practices from both agile development and UML were adopted.

## 1.5 Solution and Alternatives

As mentioned in the topics above, the final product of this work had some technology and platform requirements which, in order to enable a fast development,

limited the choosing of the type of solution and used to achieve it. Besides that, it was possible to make different approaches to the modeling of the solution, apart from the methodology, already described in the item before.

For the modeling of the solution, some simple UML diagrams were used, alongside a list of requirements. The application layout was only modelled via simple drawings that described the components of the layout and the transition between some of them, and it was later sketched in a digital format, presented later in this document.

The UML diagrams used in the project were only a use case diagram and the entity-relationship diagram for database modelling. During the project, the use of other diagrams were cogitated, like sequence diagrams or components diagrams, but after analysing the benefits they would bring, it was decided it was not worth to spend time on them. This decision is better explained in section 2.5.2.

## 1.6 Relation with Undergrad Course

This work fits the scope of the undergrad course of Control and Automation Engineering because it includes aspects of software modeling, database modeling and systems integration, while improving processes inside a company. Also, the selection of modern technologies that was made enabled the learning of programming languages not taught in the university, which is a great improvement in a person's professional life.

## 1.7 Next Chapters

On the next chapters the complete development of the application will be presented, divided into several subtopics that detail the decisions, the modeling and implementation of the solution, finishing with the results of the work. After that, the conclusions of the work will be presented, along with the future perspectives for the project.

# 2 COMPANY, PROBLEM AND SOLUTION

This chapter will describe the location in which this project was developed, what issue it was aiming to solve, and present the solution thought for the problem.

## 2.1 The Company

The project was developed inside the Secondary Air Systems (SAS) department of Rolls-Royce Deutschland, Dahlewitz site. The SAS is a subdivision of the Whole Engine Systems RRD, which is a subdivision of the System Design department.

The Secondary Air Systems department is accountable for designing, evaluating and optimising the internal and external air system of jet engines, with respect to sealing and cooling flows, bearing load and anti-icing management as well as customer and handling bleed air supply.

This project is an initiative of improving one task inside the department that is present in two different processes. The task to be improved is the comparison between air systems data from simulations and data from tests. This is part of the processes of validation of an engine and the process of predicting the impact of design modifications into air systems of an engine.

Currently, this process is done via an excel spreadsheet, in which data from three different sources (one for simulation data, one for test data and one that list the parameters available in test data) need to be copied and adapted into the file, in order for a macro in excel to compare them. After copying it, the user can visualize boxes that contain values from simulation and test data that are related, and limit deviations can be set for the spreadsheet to show, in a more intuitive way, which values are wrong.

After analysing the comparison between test and simulation data, the department can validate if the results from the tests are accurate, and if not, they will report the inaccuracies in order to get the tests fixed.

Another use of the results of the comparison are to publish the prediction of flows generated by the simulation to other departments, who can then validate the design the engine.

## 2.2 The Problem

The process described above had a series of limitations that could be solved in order to make it more efficient. A list of identified limitations is described as follow:

- The comparison between test data and simulation is currently limited for a single engine project. To make it available for different engines the spreadsheet would have to be almost completely changed.
- The fact that many people would have their own version of the spreadsheet could cause a problem of versioning, in the case somebody forgets to update the main shared file with the others.
- Because the comparison is done via a spreadsheet, it is not possible to limit the visualization of only part of the information. This means it is not possible to share the file with people outside of the department.
- The process is not available for integration with other softwares of services inside Rolls-Royce, since it is made via an excel spreadsheet. This might become a huge problem in the future, when the data from different systems inside the company should be all connected.

## 2.3 The Solution

Through the limitations shown in the previous section, it was possible to realize that there is much room for improvements in this process, so that its efficiency can be increased.

In order to solve all the problems described above, the most suitable solution would be a web application, which would not only solve the problems mentioned, but would also bring the following advantages:

- Expanded functionalities: Creating a web-application for solving the problem presented would expand the possibilities of functionalities by the use of open libraries. One example of these libraries that was used in the project is LeafletJS, a library for manipulating maps.
- Flexibility: With an increasing amount of data, Excel files get big and the performance drops. The only way of saving data within Excel are

worksheets, which need an intelligent visibility to avoid flooding the user interface. Web based technology has room for individual data handling which can be automated without letting the user know about the complexity.

### 2.3.1 Methodology

The decision of the methodology was heavy influenced by the absence of the Secondary Air Systems supervisor for this project, who wouldn't be available during the period between 26/04 and 27/06. By this reason, the period before 26/04 was focused on the creation of documents that translated the idea behind the project.

This was a decisive factor when deciding what documents to generate, because the time for understading the project was very limited. Also, this precluded the possibility of a pure agile development methodology, since after the mentioned date, it would be impossible to have regular meetings for evaluating and deciding the next steps of the project.

More details about the methodology will be detailed in section 2.5.

### 2.3.2 Technology Stack

Because the application to be developed (Engino TDM), was an extension of the application developed by the student and another intern in the previous semester (Engino), the choice of technology stack had limitations. It was possible to extend the list of libraries used for both backend and frontend, but the main languages and libraries used for the project needed to be the ones listed below:

#### React

React is a JavaScript library that divides the application into components, and only renders and updates them when they need to. This enables React applications to have an excellent performance, and also a higher productivity, because it provides a simple way for the code to be very reusable. Compared to other technologies such as AngularJS, React has a very smooth learning curve and a very easy-to-read code.

Also, it is currently the most adopted frontend framework/library, and it has the best satisfaction rate between developers who use front-end frameworks.

### Babel

Babel is a transpiler that enables people to write JavaScript code in the new ECMAScript 6 (also known as ECMAScript 2015) standard. This is a standard released in 2015 that a lot of browsers still don't support, but it enables programmers to write code faster and in a more intuitive way. Because of the lack of support by the browsers, a transpiler such as Babel is needed to convert the code into the fully supported ECMAScript 5.

### React-Router

React-Router is a library made to be used along with React. It is the responsible for handling the URL of a React web application. With it we can render specific React components according to the active URL.

### Material-UI

This is a library that implements the Material Design, which is a set of design rules for applications made by Google, into React Components. The use of this library not only makes the application much prettier and user-friendly, but also highly increases productivity, since it comes with a lot of ready-to-use React Components.

### Material-UI-Icons

This library is an implementation of google's Material Icons. It comes with a lot of React Components that render icons. The benefits are the same as the use of the Material-UI, style and productivity.

### LeafletJS

This is a pure open-source Javascript library for implementing interactive web maps. It is the responsible for all the interaction with the engine illustrations. This library is very compact and very simple to use.

React-Leaflet

React-Leaflet is responsible for implementing the LeafletJS library into React Components, providing a good integration with the rest of the code.

Phoenix

Phoenix is a backend web framework built with the language Elixir. It was selected because it provides a great improvement in performance, since Elixir compiles to the language Erlang, which was designed for high concurrency while having a small server footprint. The Phoenix framework makes the development environment very productive, allowing developers to deliver new features in a very small amount of time.

GraphQL (Apollo and Absinthe)

GraphQL is a language for making and managing queries that was implemented in this project through 2 different libraries: Apollo for the frontend and Absinthe for the backend. GraphQL helps a lot with performance issues for retrieval of complex data from databases, because it retrieves data much more efficiently than the usual Rest APIs.

PostgreSQL

PostgreSQL was chosen because it is easily integrated with Phoenix. Also, PostgreSQL can be easily compiled from its source, something that it is usually irrelevant for projects, but had a good significance for us, because of some restrictions on the servers inside the company.

## 2.4 Concepts

The concept of how the application should work can be divided into two different concepts. One is the initial concept, in which the application should be in the end of the period of the student's work. The other one is a concept for the future integration with other systems inside the company.

### 2.4.1 Initial Concept

With the application developed in this work, the process should have an improved efficiency, since it won't need to be adapted for each new file, and it will also be available for different engine projects.

The flow of the process would be changed in a way that the user would download the files from one of the three data sources mentioned in the description of the process, and then upload the file into the application, via a form. For the simulation data, this would be enough to make the data available and ready to be used for the comparison or visualization. For test files this would also be enough to enable the data to be available for comparison, but the user could also have the option to put some inputs to complement the uploaded data.

### 2.4.2 The Future

For the future, the ideal solution would be to skip the download and subsequent upload of files into the system. This means that Engino TDM should have live access to the data from the three different sources.

Even though this is the most suitable solution, because of compatibility and administration issues, this integration is something that takes a lot time to be implemented, and thus is out of the scope of this work.

## 2.5 Methodology

Because of the reasons explained in item 2.2.1, the methodology implemented was a mixture of agile methodology with standard UML, and the project was divided in two major periods, separated by the departure date of the supervisor.

Before this date, the interaction between supervisor and student followed principles of an agile methodology, while trying to generate documents from UML standards. This means that both had a very frequent interaction, so that the student could understand the problem and propose a solution, which would then be translated into UML diagrams. However, because of the short time-span for the project, an agile development was desired, and therefore only the UML documents that were judged as strictly necessary were created. These documents were the Use Case Diagram, the list of requirements, and, for the database, a conceptual ER model.

In a case where a pure agile development was implemented, the UML documents would not be created, and all the understanding of the application would be made by the interactions between supervisor and student. This would enable the project to be focused in the implementation, and consequently more of the application would be ready by the end of the working period. However, this approach was impossible to be implemented as a whole, for the reasons already presented.

The opposite approach would be the implementation following the UML standards, with the creation of many diagrams to describe the application. However, this approach is something suitable for teams with many members and without a close interaction with the clients, because in this case the UML diagrams would enable the team to have a common understanding of the problem, and present the idea of the concept to the client in a more understandable way.

Since this project was developed by one person with the consultancy of another, the methodology aimed was the agile development. However, because of the time limitation of supervision, the mixed approach was chosen.

# 3 MODELLING

In this chapter all the steps of the modelling of the application will be presented, starting with the list of tasks needed to be done in order to complete the application, then presenting the time division for each task, the modelling UML specifications, layout and the database.

## 3.1 Tasks

In order to complete the modelling, planning and implementation of the application, a list of tasks, that was later converted into a schedule, was created, and the items in it, with their earlier descriptions, are presented below:

**Study of the Problem:** The first step before starting the development of the system is the study of the problem, in order to understand what the application is solving, where the application fits and how is this task currently done.

**General View of the System:** As Wazlawick (2004) suggests, in the beggining of a software project, it is interesting to have a text that shortly describes the application. This helps to understand the main goal of the project and present the idea to other stakeholders.

**Use Case Diagram:** Creation of a simple use case diagram, envisioning making the task of raising the software requirements easier.

**Requirements:** The main document of the early-phase of the project. Lists all the requirements for the software in a table-shaped document.

**Layout:** The creation of the layout of the application, showing the screens and transition between them, should be made via simple drawings and drafts to show the behavior of layout components.

**Further UML Diagrams:** The creation of UML diagrams other than the Use Case Diagram.

**Database Modelling:** Modeling of the database using Relational Entities.

**Review of Documentation:** Period of time used to review the documents created.

**Creation of Database:** The database that will be used is PostgreSQL, and it will be created using the framework Phoenix, whose development language is Elixir.

**Queries in the Backend:** In order to retrieve the data using the Elixir language, some queries must be created. This will be done using the library Absinthe, which is an implementation of GraphQL for Elixir.

**Processing Scripts:** Since the data that will feed the database is currently in text documents and spreadsheets, there is a need to create processing scripts that can parse the data and then store it in the database. These scripts should be created using the language Elixir.

**Front-end Implementation:** Will be made using the libraries ReactJS, Redux, React-Router and Material-UI.

**Queries in the Frontend:** In order for the Frontend to treat the returned data, it will be necessary to implement the queries in the frontend. This will be done using the library Apollo, which is an implementation of GraphQL for React.

**Integration with Other Systems:** A verification of the possibility to integrate the application with the other systems that are data sources should be done.

**Review of the Application:** Realize tests with the application to check if it meets the requirements and raise bugfixes.

**Bugfixes:** Fix the bugs raised in the previous item.

**Documentation:** Creation of documents describing how Engino TDM should be used and describing the application and its code, enabling future developers to change the code easily.

## 3.2 Schedule

After defining the necessary steps to develop the application, a schedule for the project was created. The schedule was focused on the creation of documents until April 26th, since during this period the student would still have supervision. After this date, the schedule was focused on development. The images below show the division of the tasks on time.

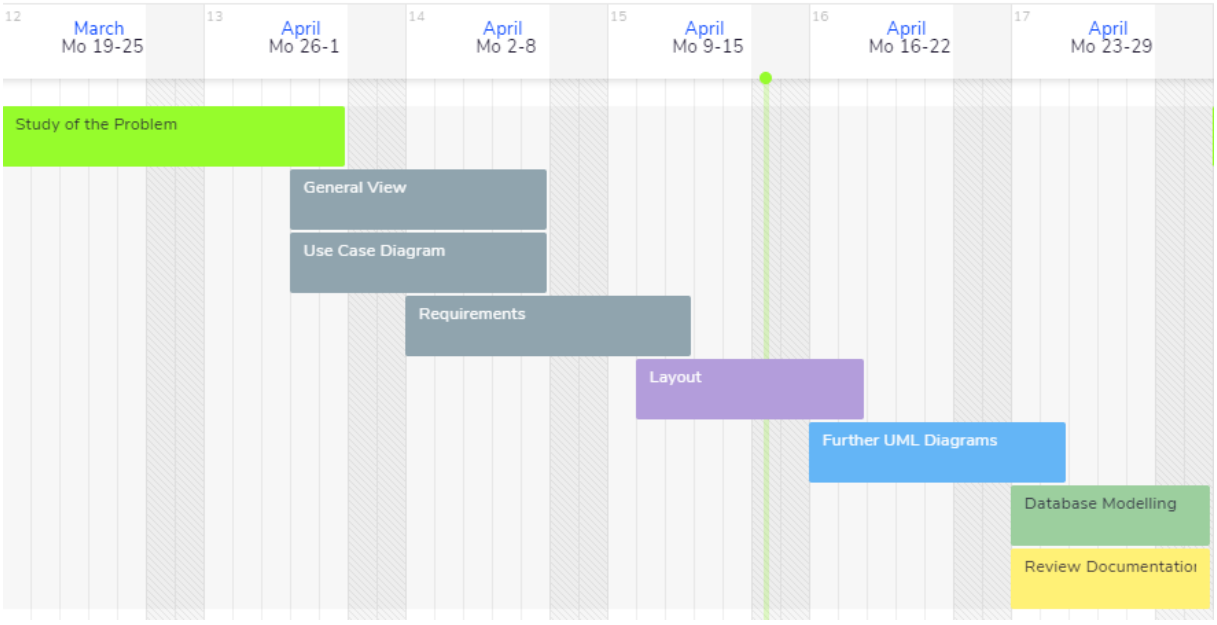


*Figure 1: Schedule March/April*

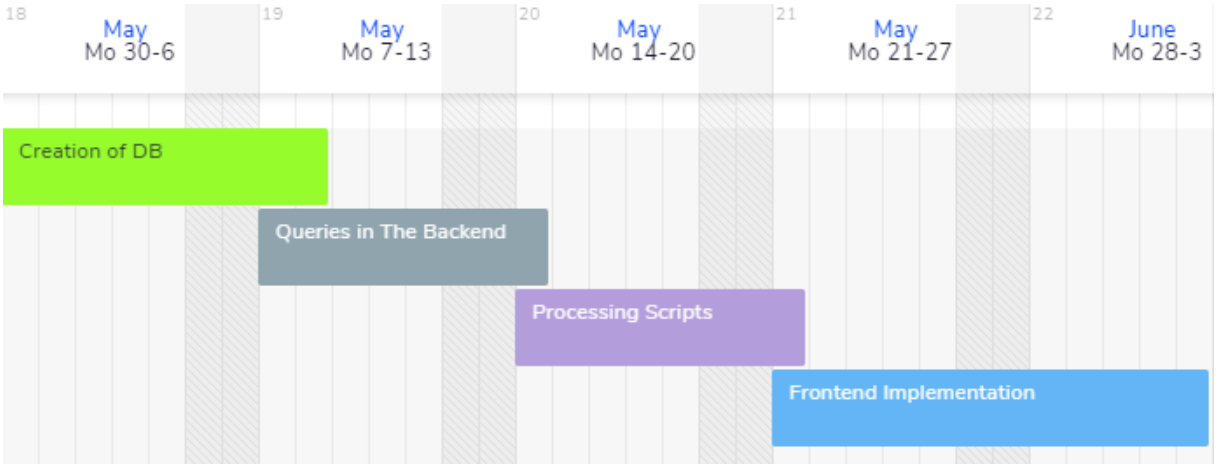


*Figure 2: Schedule May/June*

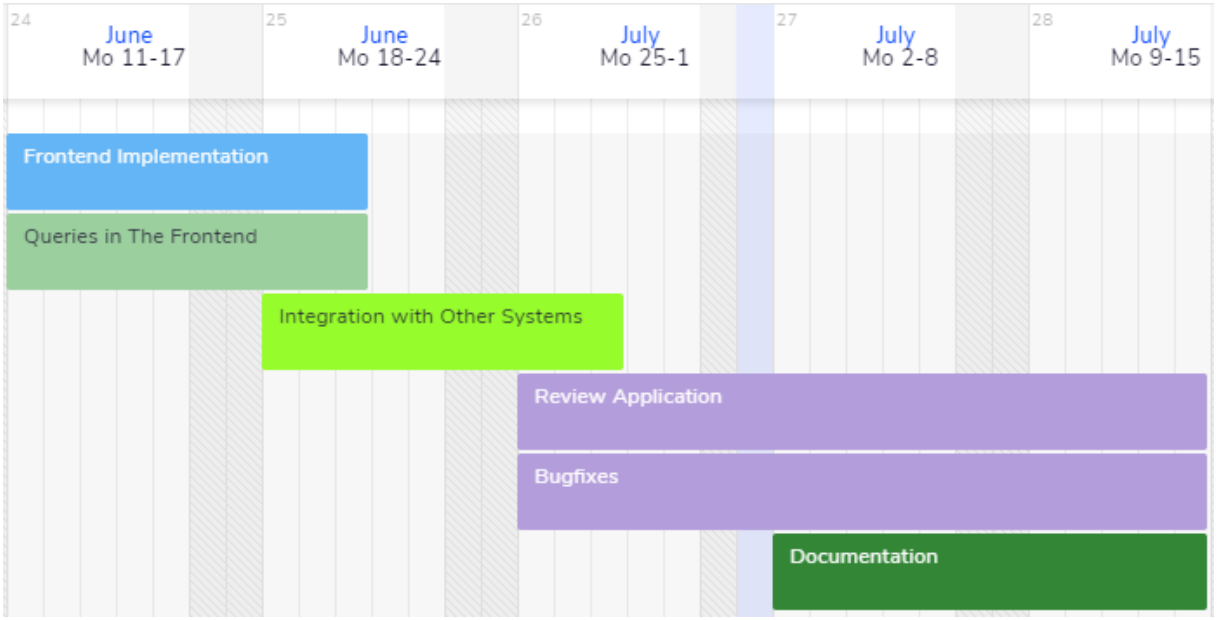


*Figure 3: Schedule June/July*

Along the development, changes in the schedule needed to be done. The two greatest motives behind the changes were due to the need to cooperate in other projects, and due to the complexity of some tasks that were not initially predicted.

For these reasons, the effective time division was different from the one predicted by the schedule, and one of the main differences was the absence of further UML diagrams for the application, apart from the ER diagram that was used later for the database modelling. The decision of not having further UML diagrams was made after an evaluation of the remaining time to complete the modelling. As the time was growing short, the need to have a good model of the database seemed more important.

Apart from that, a list of possible UML diagrams were evaluated, and none showed a good cost-benefit relationship.

The two most likely UML diagrams to be implemented were the components diagram and the sequence diagram. However, a sequence diagram would not make much sense in a single-page application, in which the user can navigate through all the application as he wishes and the components are completely dynamic. On the other hand, a component diagram could be used to represent all the components in the application, however, the amount of components would bring a great difficulty in modelling this kind of diagram, and in the end it would not be used, since the division in components that can be made using React makes is already sufficient and more intuitive to understand the code in the future.

Another change in the schedule was in the division between the implementation of backend and frontend, which was not so separated. In the early stages of the application, the backend code was made alltogether, before implementing the frontend. However, in order to prevent creating unnecessary parts of the code, the rest of the backend was created “on the go”. This means that parts of the backend, such as complex queries in the database, were created only in the moment they were needed.

The last difference from the original schedule was the evaluation of the possibility to integrate with other systems. This task was postponed for the period after the deliver of this document. This was decided after the difficulties in the implementation appeared, as it was seen that the project would need to be extended. Moreover, this is a task that is dependant on other departments, and in order for being able to implement this, a lot of bureaucracy would have to be surpassed.

### 3.3 System General View

The first outcome of the project was the General View of it. According to Wazlawick (2004), the general view of a system is a text without a specific structure that describes the main ideas about a system. For the project, the general view created was the following:

## Engino TDM General View

"Engino TDM is a web application focused on the visualization and comparison of secondary air systems data from Rolls-Royce Deutschland, Dahlewitz site. The application aims to make the process of generating data overviews and validating test data faster and more intuitive.

The data that will be presented are simulation data, originated in the software SPAN (Suite of Programs for Air Networks) and stored as formatted text files; and data from engine tests, which are stored as excel spreadsheets.

The system has 3 main functionalities, which are:

- Comparison of data from SPAN and measured engine data via an interactive technical illustration that gives the user the ability to zoom in or out and move through these 2D drawings of the engines, showing pressure and temperature data in specific points of the engine (nodes).
- The visualization and comparison of data from SPAN, which also uses interactive technical illustrations, but with an additional functionality of visualizing the air flows of a system
- The comparison of data from various engines, through a plot that can be accessed by clicking in one of the points from the previous functionalities, or can be manually configured by the user to show the data he wants

In order for the system to achieve these 3 main functionalities, it needs to have a lot of multiple smaller functionalities that together make the system behave as expected. Some of these functionalities are the selection of which datasets (from test data and SPAN) are going to be used for the interactive illustrations; selection of which parameters are going to be plotted; the possibility to add nodes on the comparison of SPAN and test data; the possibility to upload SPAN files, test files, and instrumentation lists; possibility to visualize, edit and delete the available datasets (test and SPAN files) and instrumentation list; validation of the difference between SPAN and test data to check if it is among user-defined limits; selection of which parameters are going to be used to calculate the pressure and temperature average values presented in nodes with test data; possibility to select which parameters are going to be used to calculate

the scaled and predicted values of SPAN data; and the possibility to convert the values into different measurement units.”

### 3.4 Use Case Diagram and Requirements

After creating the text for the General View, a simple Use Case Diagram was created, in order to facilitate the process of creating requirements. A Use Case Diagram represents the main actions and interactions that users of the system can make. Actors are the different types of users that are going to use the system, and the cross-reference table associates which of the requirements are associated with each use case.

The system has 3 types of actors, separated into Super Users, Special Users and Basic Users. The Basic user only has capabilities to visualize the data and to change the application state, but not the database. Meanwhile, the Special Users have access to database, while having the same basic privileges as the basic users. The Super Users have all the capabilities of the special users and can also manage users.

The image below shows the use cases and the relationship between the actors, in which, for visualization purposes, the use cases are only related to their lowest level type of user.

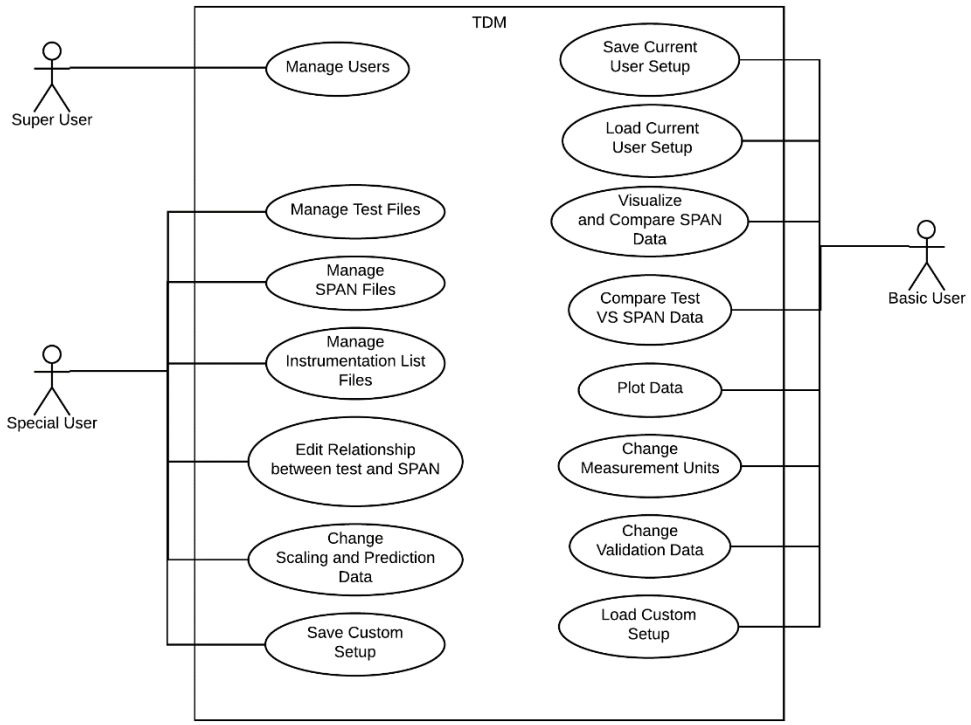


*Figure 4: Use-Case Diagram*

The following table shows the description of each use case along with the cross-reference between them and the requirements, which can be found in Appendix A. The actors are representing by their acronyms: Super User (SU), Special User (SPU), Basic User (BU).

| Name | Actors | Description | Cross-Reference |
|---|---|---|---|
| Manage Users | SU | Creating, deleting or editing the users of the system | F18, F19, F20 |
| Manage Test files | SU, SPU | Uploading, editing and deleting test files | F2, F5 |
| Manage SPAN files | SU, SPU | Uploading, editing and deleting SPAN files | F1, F4 |
| Manage Instrumentation list files | SU, SPU | Uploading, editing, deleting and changing the content of instrumentation list files | F3, F6, F9 |
| Edit relationship between test and SPAN | SU, SPU | Set the relationship between the test parameters and SPAN nodes | F12, F30, F31 |
| Change Scaling and Prediction Data | SU, SPU | Change the parameters used to calculate the scaled and predicted values of SPAN | F13, F14, F15 |
| Save Custom Setup | SU, SPU | Save the current state of the program (which datasets are selected, what parameters are plotted, etc) | F8 |

| | | | |
|---|---|---|---|
| Load Custom Setup | SU, SPU, BU | Load the saved state of the program (selected datasets, plot configuration, etc) | F9 |
| Visualize and compare SPAN Data | SU, SPU, BU | Select and compare SPAN data from different runs | F14, F15, F25, F26, F27, F28 |
| Compare Test VS SPAN Data | SU, SPU, BU | Select and visualize the comparison between SPAN and test data | F14, F15, F16, F17, F29 |
| Plot Data | SU, SPU, BU | Select parameters and plot them | F22, F21 |
| Change Measurement units | SU, SPU, BU | Change the measurement units used in the system | F23 |
| Change validation data | SU, SPU, BU | Change which data is used to calculate the parameter average values for test data | F24, F32 |
| Save Current User Setup | SU, SPU, BU | Save the current state of the program (which datasets are selected, what parameters are plotted, etc) | F10 |
| Load Current User Setup | SU, SPU, BU | Load the saved state of the program (selected datasets, plot configuration, etc) | F11 |

Using the definition by Raul Sidnei Wazlawick, a document of requirements is a document that contains all the functionalities of the system. The requirements can be divided into two categories:

- **Functional requirements (F),** which list everything the system should do
- **Non-functional requirements (NF),** which represent restrictions about how the system should do its functional requirements
- **Supplementary Requirements (S),** that are applied to the system as a whole, not only to specific functions

Functional requirements can be divided into:

- **Plain Functional Requirements**, that are performed with the perception of the user. This requirements are usually events and responses of the system, that is, any exchange of information between the interface and the external environment
- **Hidden Functional Requirements**, that are performed without the explicit perception of the user.

Because of the size of the list, the complete table of requirements is shown in the Appendix A of this document.

### 3.5 Layout Definition

After the complete understanding of the problem and an extensive analysis of the requirements, a layout definition was built. This layout was validated by the supervisor from Secondary Air Systems department.

The layout proposed is presented below, with a description of the image after each one.

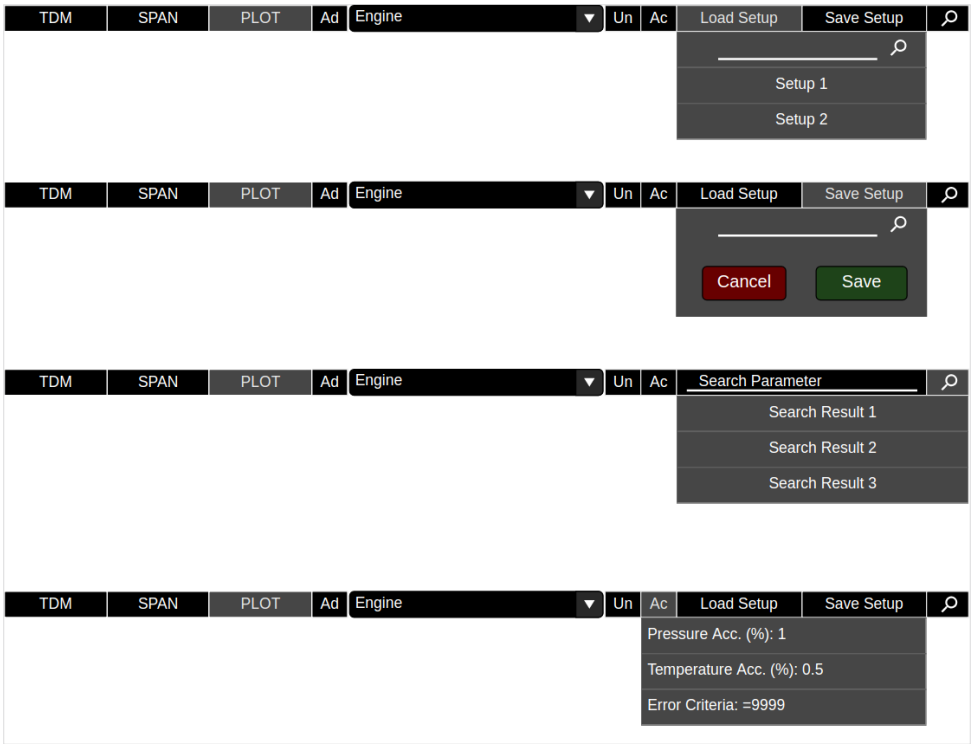


*Figure 5: Layout Navigation Bar*

This first picture shows the navigation bar of the application. It has the functionality of changing between the three main screens (TDM, SPAN and Plot), loading and saving different setups, searching for nodes or flows, changing the accuracy criteria, changing the measurement units and selecting different engines. It also contains a link to the admin page, which is shared with the Engino application.

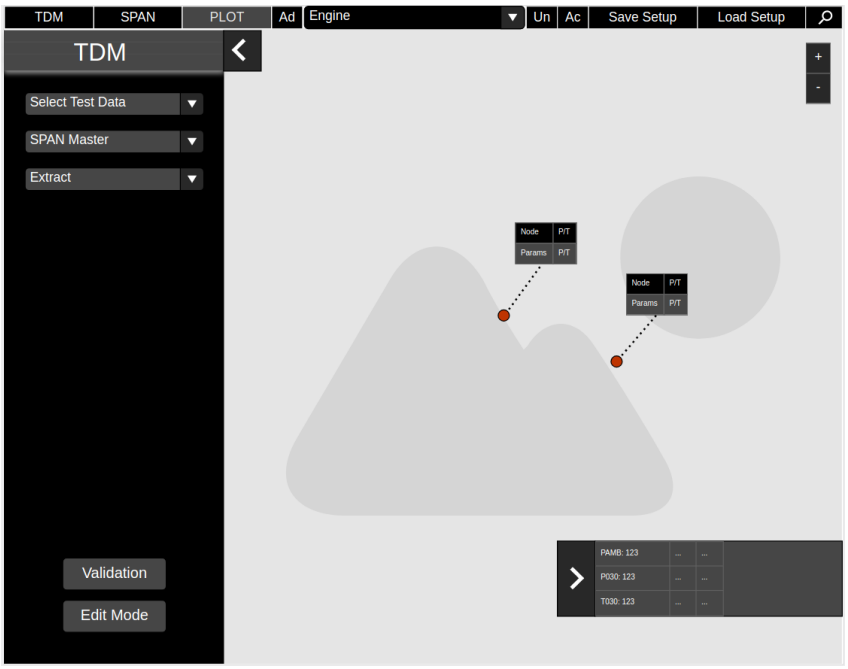


*Figure 6: Layout TDM Screen*

The picture above shows the main screen of the application, whose functionality is to compare nodes from simulation (SPAN) data and parameters measured during tests.

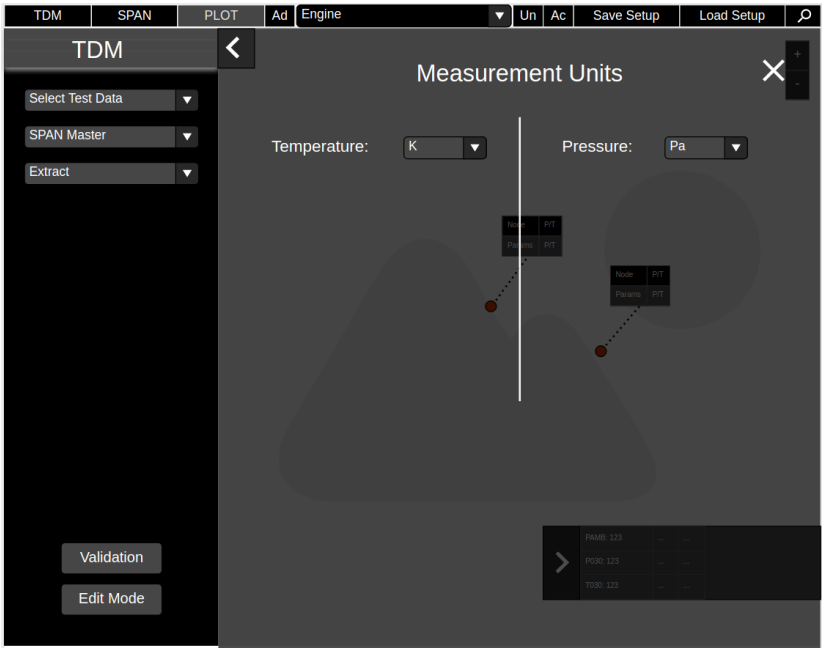


*Figure 7: Layout TDM Measurement Units*

The image above depicts how the measurement units should be presented, via a translucid dialog with the selection options for units that the user can make.

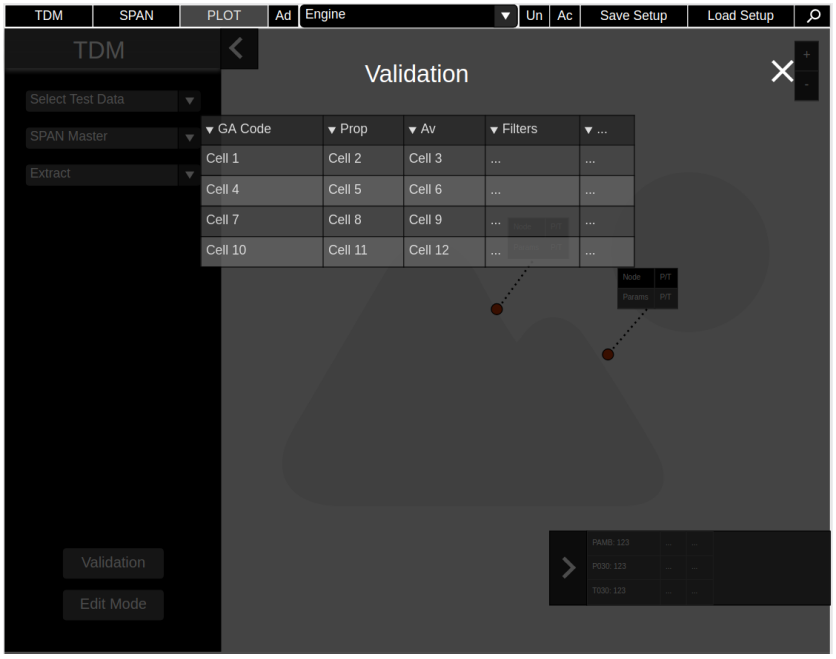


*Figure 8: Layout TDM Validation*

The validation screen shows which parameters are going to be used in the comparison. This is necessary because some parameters are actually the average of multiple parameters, and in some cases, an isolated parameter can have an undesired influence in the average.

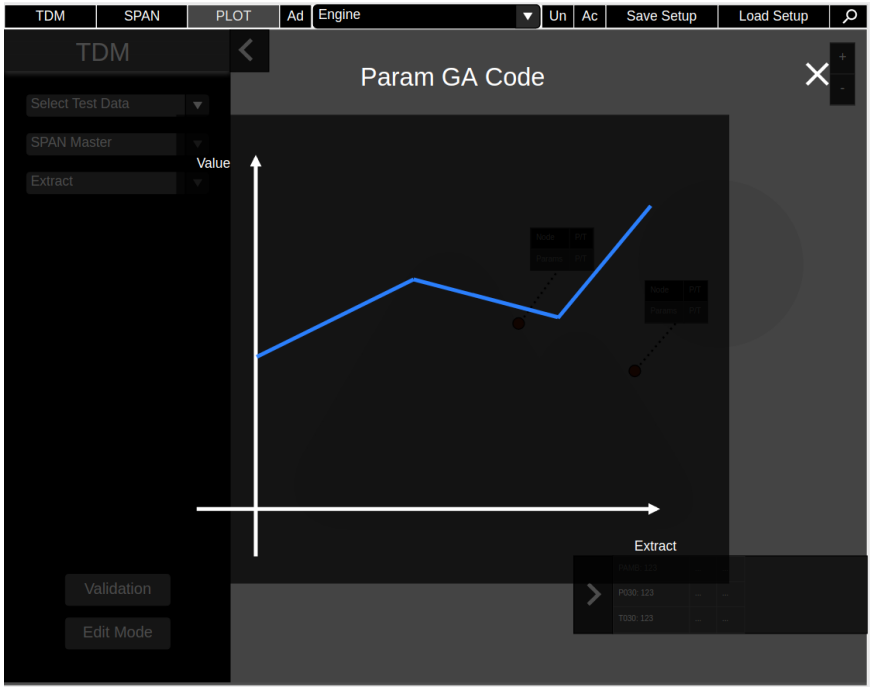


*Figure 9: Layout Validation Graph*

In the validation screen, the user should be able to click on a parameter to open a graph showing the information about that parameter in time (in different extracts).

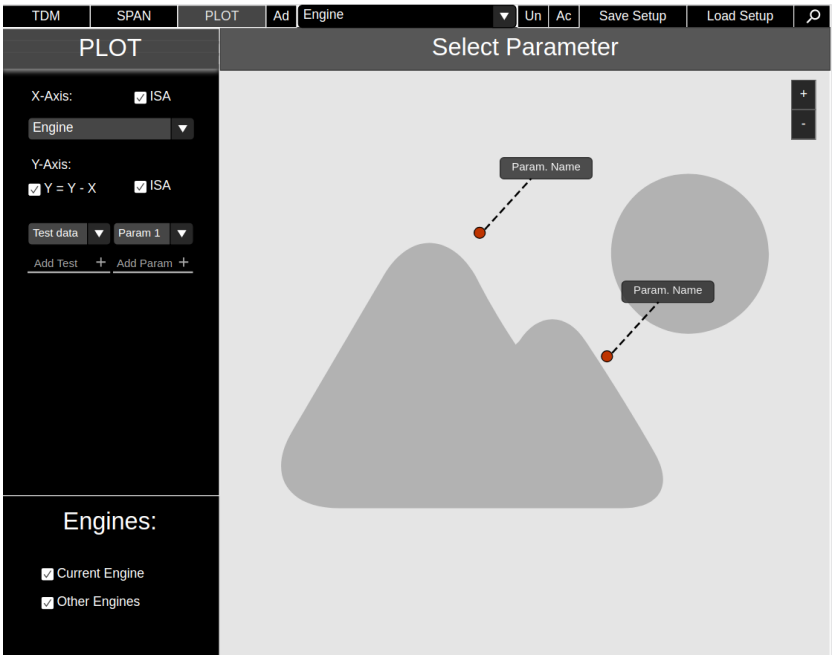


*Figure 10: Layout Plot Screen – Parameter Selection*

The plot screen is a general screen in which the user should be able to select the parameters that he wants to plot, being able to compare different parameters from different test data, and also parameters from different engines. This first image (above) shows how the user should select the parameters, via a map that show the available parameters for the selected test data. The next image will show a representation of the graph itself.


*Figure 11: Layout Plot Screen*

The last one, the SPAN screen, shows the information related to the simulation data. In this screen the user can visualize nodes and flows and compare them via tables of graphs.

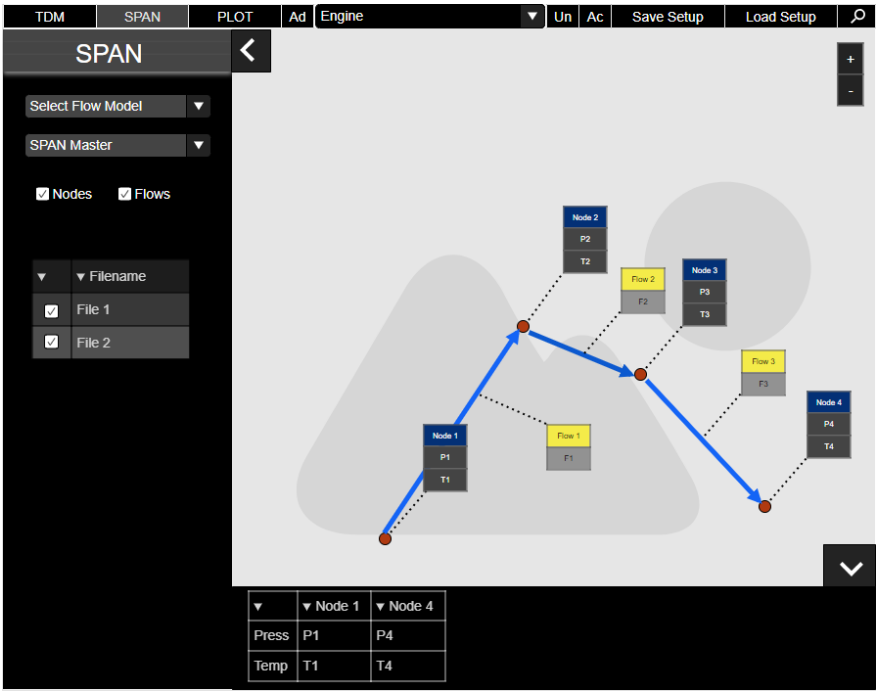


*Figure 12: Layout SPAN Screen*

### 3.6 Database

The database modelling started by the creation of a list of data elements, and then a list of assumptions related to the data that would be stored in the database. These assumptions gathered the correlation between different types of data, and it was essential for the creation of the ER model that will be presented afterwards.

The essential types of data listed were:

- Engine Type: The project of an engine, also called Engine Common Name
- Engine Number: One version of the engine project
- Engine Build: One of the builds of an engine
- Engine View: One visual representation of an Engine Type
- Test file: The file that contains test data
- Test Parameters: An information about one measurement location in an engine
- Extracts: The instants in time when test data was collected
- SPAN file: A simulation file

- Flow: The definition of a flow in a simulation file
- Flow Points: The points which a flow contain
- Nodes: Points in an engine where simulation data was calculated
- Instrumentation List: List containing all information about instrumentation
- ERF: Experiment Request Form, a document that stablishes what are the requirements for an experiment.
- SLN: Document that declares a solution for a raised issue.

The list of assumptions is the following:

- Test parameters can be in multiple engine types.
- Each engine type contains multiple engine numbers, but one engine number can only be associated with only one engine type, even though it is possible to have equal engine numbers across different engine types.
- Each engine number contains multiple engine builds, but one engine build is always associated to only one engine number, even though it is possible to have equal engine builds across different engine numbers.
- Each test file is associated to a single engine build, but an engine build contains multiple test files.
- Each test file contains multiple parameters, and each parameter can be associated to multiple test files.
- Test parameter names are unique. OBS: Test parameters can repeat for different engines. This means that the same name can appear in different engines.In this case, they will be considered different parameters, because even though they have the same functionality, they will probably have different positions on the technical drawing.
- Each engine type contains multiple SPAN files, but each SPAN file is associated with a single engine type.
- Each SPAN file has multiple flows, and each flow can be in multiple SPAN files.
- Each SPAN file has multiple nodes, and each node can be in multiple SPAN files.
- Each instrumentation list is associated with only one engine type, and each engine type contains only one instrumentation list.

- Each instrumentation list contains multiple parameters, but one parameter can be in a single instrumentation list (considering that parameters with the same name in different engines are different parameters).
- Each flow is associated with one starting node and one end node, but nodes can be associated with multiple flows.
- Each flow is associated with multiple flow points, but one flow point is always associated to one unique flow.
- One ERF can be associated with multiple engine builds, and each engine build can have multiple ERFs.
- Each ERF contains multiple parameters, and each parameter can be in multiple ERFs.
- The combination between ERF, Engine Build and Parameter is unique.
- Each combination of ERF, engine build and parameter can be associated with multiple SLNs, and one SLN can be associated with multiple combinations.

With this list of assumptions, a conceptual Entity-Relationship model was created, following the concepts presented by Heuser (2009). This model is presented in the image below. In it, the attributes were hidden, in order to improve the visualization.

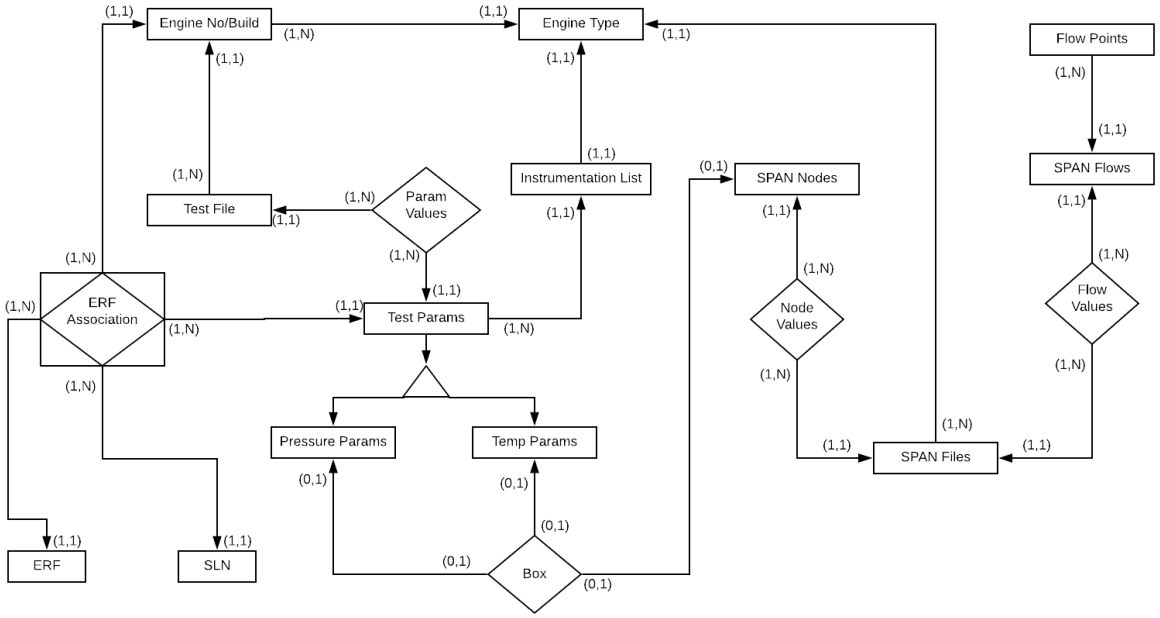


*Figure 13: Conceptual Entity-Relationship Model*

With the ER model, it was possible to create the database, also using the methodology explained in Heuser (2009).

The final model of the database is presented below. For the creation of the final version some minor changes were made. Also, the database for users was not included in the modelling because the application will share the same database as the Engino application.

In the picture below the tables of the database with its relationships are shown. In each table, the first column shows the type of each database table column, the second shows the column name, and the last one depicts which of the columns are primary keys (P) and foreign keys (F).

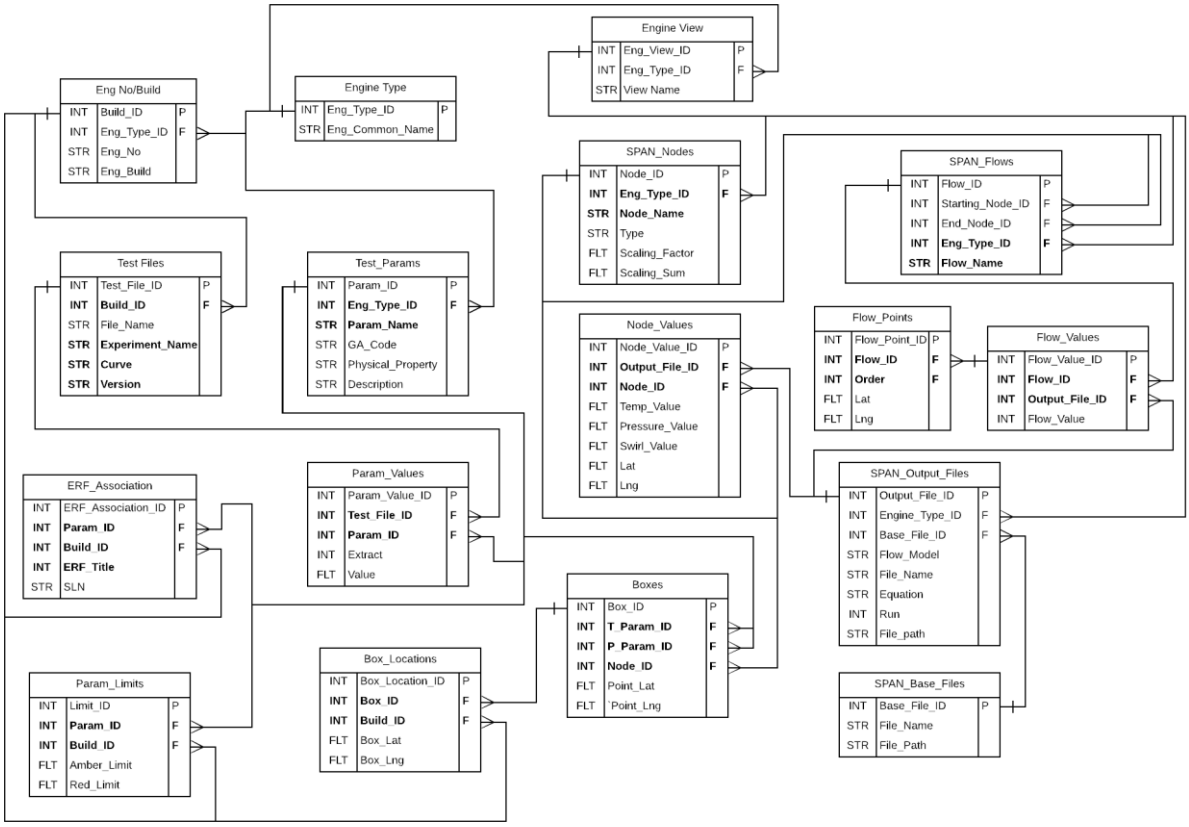


*Figure 14: Database*

From the tables shown in the image above, the tables Engine No/Build, Engine Type and Engine View are from Engino database, so this ones were not created.

# 4 IMPLEMENTATION

This chapter will present the implementation of the application modelled, a step that started after the modelling of the database. The first task was to translate the database into the migrations used by the Absinthe library, which is an implementation of GraphQL for Elixir. The migrations are translated into database creation or updating commands later when the user executes the migration command in the command line. The image below shows an example of a mutation. This one is responsible for creating the table for the output files.

```
def change do
  create table(:output_files) do
    add :file_name, :string
    add :flow_model, :string
    add :equation, :string
    add :run, :string

    add :engine_common_name_id, references(:engine_common_names, on_delete: :nothing), null: false
    add :base_file_id, references(:base_files)

    timestamps()
  end
end
```

*Figure 15: Mutation*

After the creation of the migrations, it was time for the creation of the schemas, which are the representation of data that will be available for querieng in the frontend. In this step only the basic schemas and schema types were created, alongside simple queries. As mentioned in the 2.5.2 section, the complex queries were implemented later, as they were needed. The image below shows the type definition for the output files, defined in Elixir.

```
object :output_file do
  field :id, :integer
  field :file_name, :string
  field :flow_model, :string
  field :equation, :string
  field :run, :string

  field :engine_common_name, :engine_common_name, resolve: dataloader(Engino.Engine)
  field :base_file, :base_file, resolve: dataloader(Engino.SAS)
  field :node_values, list_of(:node_value), resolve: assoc(:node_values)
  field :flow_values, list_of(:flow_value), resolve: assoc(:flow_values)
end
```

*Figure 16: Schema Type*

In the next image a simple query is presented, defined as a field. This means that by defining this query, we will have the output files available as simple GraphQL fields.

```
@desc "Get a single Span Output File by ID"
field :output_file, :output_file do
  arg :id, :integer

  resolve &SpanOutputFilesResolver.get/2
end
```

*Figure 17: Simple Query*

In order to have a good separation of the Engino code from the Engino TDM backend, a different context in the Phoenix Framework, which are used to separate parts of the code with different functions.

After the creation of the database and the main queries in the backend, the implementation of the frontend started, in a cyclic implementation which the frontend was made, and then, if complex queries were needed, they were made in the backend also.

The image below shows and example of the definition of a complex query in the schema.

```
@desc "Get a list of output files with one of the IDs and return only the flows in the array"
field :output_files_by_ids_and_flows, list_of(:output_file) do
  arg :file_ids, list_of(:integer)
  arg :flow_ids, list_of(:integer)

  resolve fn %{file_ids: file_ids, flow_ids: flow_ids}, _ ->
    {:ok, Engino.SAS.list_output_files_by_ids_and_flows(file_ids, flow_ids)}
  end
end
```

*Figure 18: Complex Query*

The next image shows the function that was called to resolve this query, which needed to perform a join between different tables in the database.

```
def list_output_files_by_ids_and_flows(file_ids, flow_ids) do
  Repo.all from of in OutputFile,
    join: fv in assoc(of, :flow_values),
    where: of.id in ^file_ids,
    where: fv.span_flow_id in ^flow_ids,
    preload: [flow_values: fv],
    select: of
end
```

*Figure 19: Query Resolver*

The frontend was implemented using the library React, alongside Redux and React-Router, which are libraries for handling the states of the application and the url of it, respectively. Also, the frontend used the library LeafletJS, in its React implementation, called React-Leaflet, which is an open library for dealing with interactive maps, and used also the Material-UI library, which implements the Google Material Design principles into react components.

In the next section, the results of this work will be presented, including images of the implementation of the application.

# 5 RESULTS

The work described in this document had as main result the complete modelling of the application that solves the issues raised by the Secondary Air Systems Department.

Also, an initial version with some basic functionalities was developed during the period of this work. This initial version of the application already has some core functionalities which enables the use of it for improving the efficiency of the department's processes.

In the following images it is possible to see what parts of the application are already made. For export control reasons, the information in the application was replaced by constant numbers and random positioning, and the image was replaced by an image of a jet engine found on the internet.

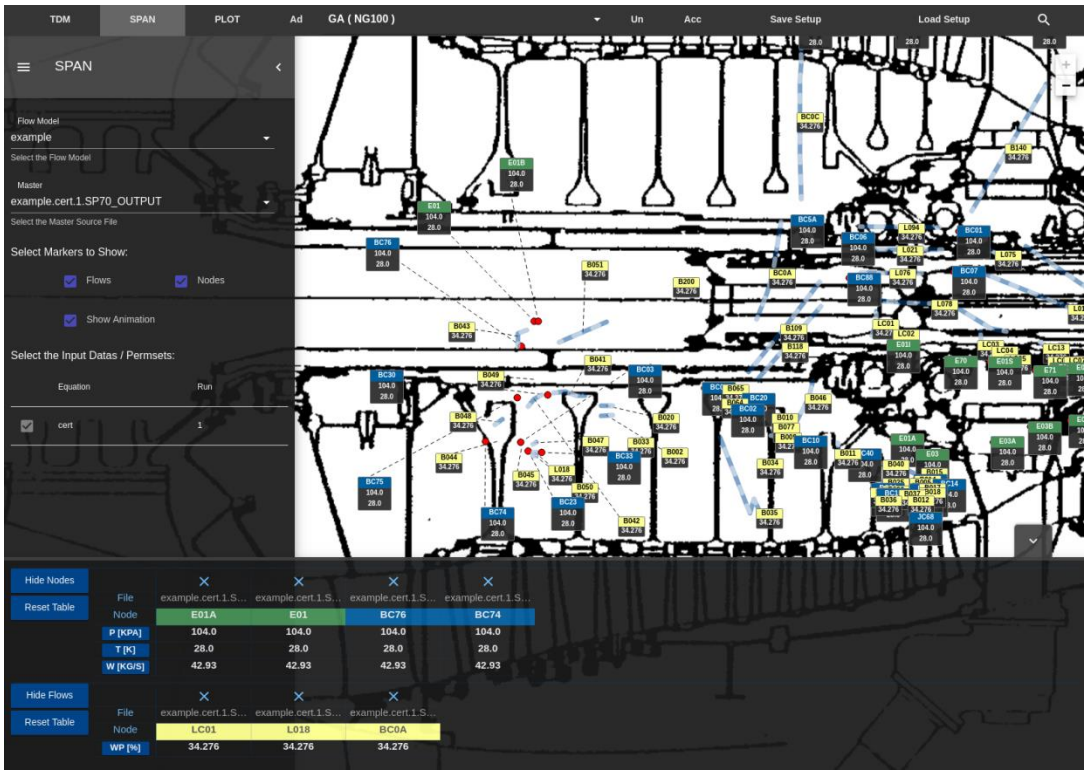


*Figure 20: SPAN Screen*


*Figure 21: SPAN Screen with Graph*

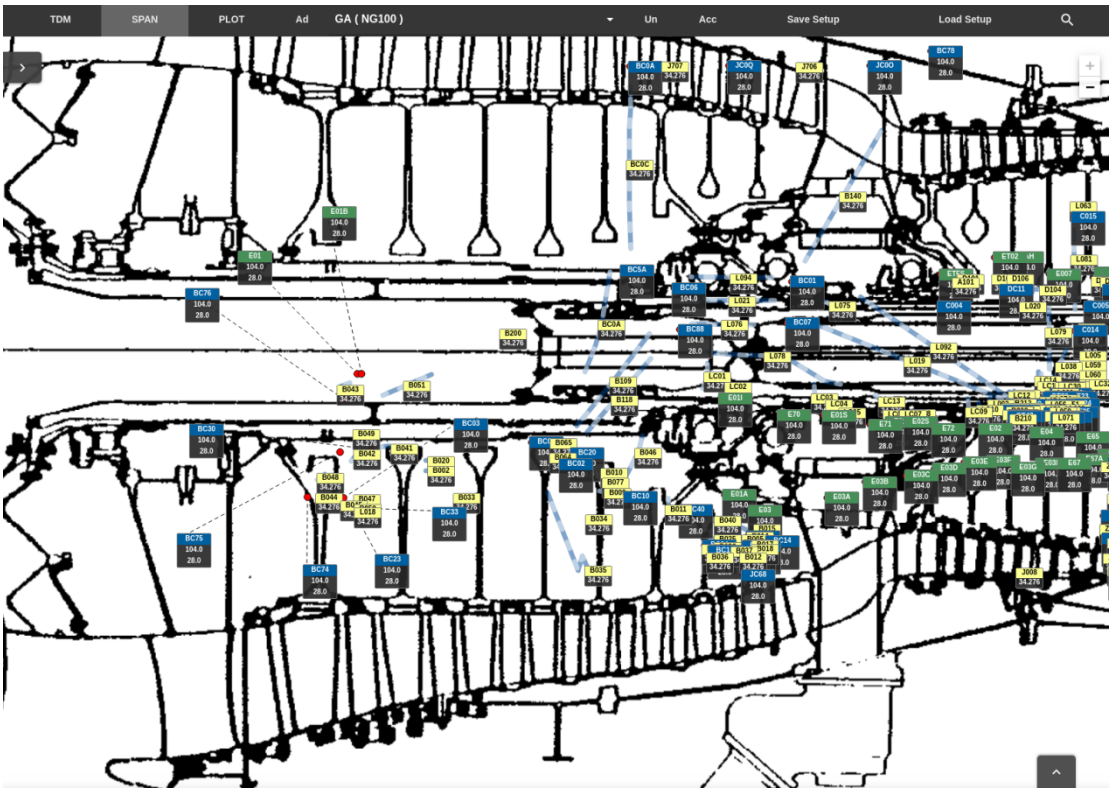


*Figure 22: SPAN Screen with Sidebar and Bottombar collapsed*

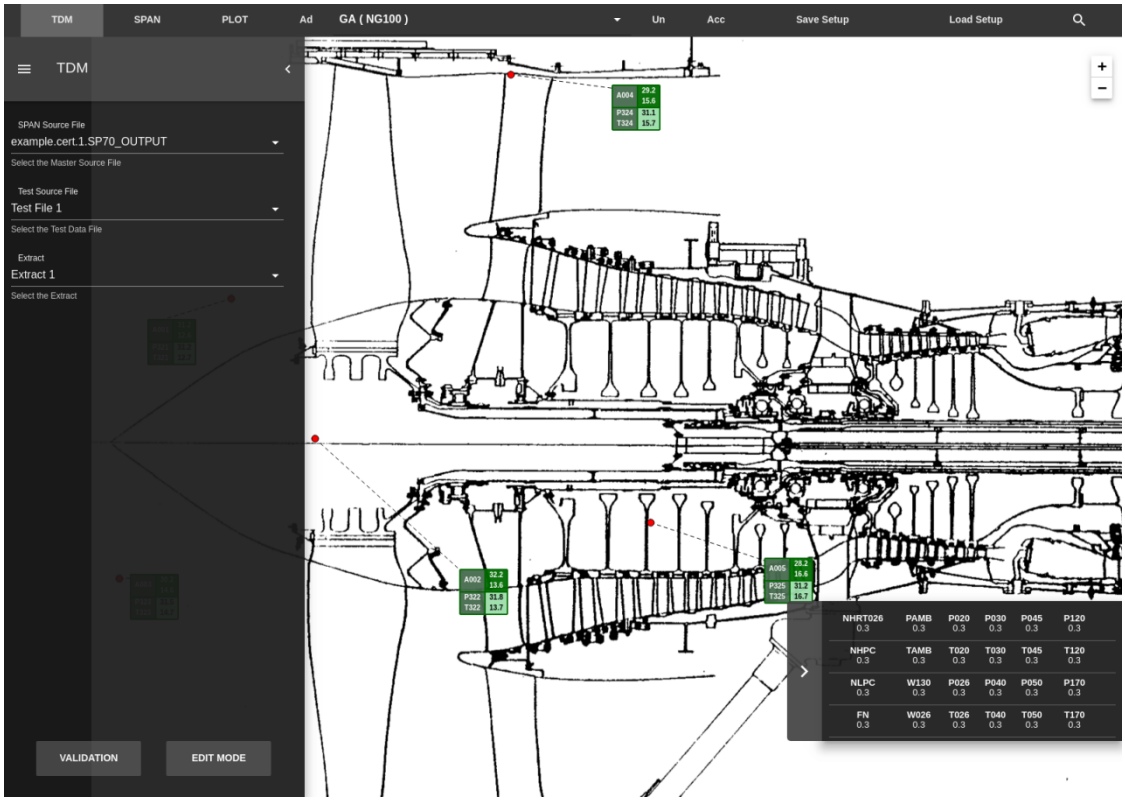


*Figure 23: TDM Screen*

## 5.1 Verification

In order to verifiy that the application was working and met the requirements for it, some initiatives were made. The first was a verification of the consistency of the database and of the simple queries required to have the application working properly. After that, a usability proof was done, in order to check what was achieved, and then this was compared to the requirements raised in section 3.4, available in Appendix A.

### 5.1.1 Database check

In order to check that the database was created correctly and had the correct relationship between tables, a set of “dummy” data was inserted in it. This data was stored via a seeds file, a file that can be programmed to insert information in the database via implemented backend functions on Elixir. The image below shows part of the seeds file used to create this dataset.

```
# Seed SPAN Files
{:ok, base_file} = Engino.SAS.create_base_file %{file_name: "SPAN Base File 1"}
{:ok, output_file} = Engino.SAS.create_output_file %{file_name: "SPAN Output 1", equation: "Cert", flow_model: "FM 1", run: "1", base_file_id: base_file.id, engine_view_id: engine_view.id}
# Seed SPAN Nodes
{:ok, span_node1} = Engino.SAS.create_span_node %{node_name: "N001", type: "BO", engine_view_id: engine_view.id}
{:ok, span_node2} = Engino.SAS.create_span_node %{node_name: "N002", type: "VE", engine_view_id: engine_view.id}
{:ok, span_node3} = Engino.SAS.create_span_node %{node_name: "N003", type: "IN", engine_view_id: engine_view.id}
{:ok, span_node4} = Engino.SAS.create_span_node %{node_name: "N004", type: "IN", engine_view_id: engine_view.id}
{:ok, span_node5} = Engino.SAS.create_span_node %{node_name: "N005", type: "IN", engine_view_id: engine_view.id}
{:ok, span_node6} = Engino.SAS.create_span_node %{node_name: "N006", type: "IN", engine_view_id: engine_view.id}
# Seed SPAN Node Values
Engino.SAS.create_node_value %{span_node_id: span_node1.id, output_file_id: output_file.id, lat: -10.875, lng: 56.5, pressure_value: 13220.21, temp_value: 425.05 , swirl_value: 343.03}
Engino.SAS.create_node_value %{span_node_id: span_node2.id, output_file_id: output_file.id, lat: -39.515625, lng: 20.796875, pressure_value: 1745.05, temp_value: 152.34 , swirl_value: 745.34}
Engino.SAS.create_node_value %{span_node_id: span_node3.id, output_file_id: output_file.id, lat: -92, lng: 97.8600006, pressure_value: 5234.53, temp_value: 343.43 , swirl_value: 256.43}
Engino.SAS.create_node_value %{span_node_id: span_node4.id, output_file_id: output_file.id, lat: -32, lng: 65.76, pressure_value: 234.53, temp_value: 2435.43 , swirl_value: 534.43}
Engino.SAS.create_node_value %{span_node_id: span_node5.id, output_file_id: output_file.id, lat: -55, lng: 127, pressure_value: 775.53, temp_value: 2434.43 , swirl_value: 34.43}
Engino.SAS.create_node_value %{span_node_id: span_node6.id, output_file_id: output_file.id, lat: -12, lng: 126, pressure_value: 3455.53, temp_value: 24.43 , swirl_value: 123.43}
# Seed SPAN Flows
{:ok, span_flow1} = Engino.SAS.create_span_flow %{flow_name: "F001", engine_view_id: engine_view.id, starting_node_id: span_node1.id, end_node_id: span_node2.id}
{:ok, span_flow2} = Engino.SAS.create_span_flow %{flow_name: "F002", engine_view_id: engine_view.id, starting_node_id: span_node2.id, end_node_id: span_node3.id}
{:ok, span_flow3} = Engino.SAS.create_span_flow %{flow_name: "F003", engine_view_id: engine_view.id, starting_node_id: span_node3.id, end_node_id: span_node6.id}
{:ok, span_flow4} = Engino.SAS.create_span_flow %{flow_name: "F004", engine_view_id: engine_view.id, starting_node_id: span_node1.id, end_node_id: span_node6.id}
# Seed SPAN Flow Values
{:ok, flow_value1} = Engino.SAS.create_flow_value %{span_flow_id: span_flow1.id, output_file_id: output_file.id, value: 1232.213}
{:ok, flow_value2} = Engino.SAS.create_flow_value %{span_flow_id: span_flow2.id, output_file_id: output_file.id, value: 5632.213}
{:ok, flow_value3} = Engino.SAS.create_flow_value %{span_flow_id: span_flow3.id, output_file_id: output_file.id, value: 2413.213}
{:ok, flow_value4} = Engino.SAS.create_flow_value %{span_flow_id: span_flow4.id, output_file_id: output_file.id, value: 41234.213}
# Seed SPAN Flow Points
Engino.SAS.create_flow_point %{lat: -20.032, lng: 45.23, order: 1, flow_value_id: flow_value1.id}
Engino.SAS.create_flow_point %{lat: -25.032, lng: 48.76, order: 2, flow_value_id: flow_value1.id}
Engino.SAS.create_flow_point %{lat: -67.45, lng: 55.45, order: 1, flow_value_id: flow_value2.id}
Engino.SAS.create_flow_point %{lat: -67.45, lng: 55.45, order: 1, flow_value_id: flow_value3.id}
Engino.SAS.create_flow_point %{lat: -54.45, lng: 50.45, order: 2, flow_value_id: flow_value3.id}
Engino.SAS.create_flow_point %{lat: -45.45, lng: 45.45, order: 3, flow_value_id: flow_value3.id}
Engino.SAS.create_flow_point %{lat: -12.45, lng: 66.45, order: 1, flow_value_id: flow_value4.id}
Engino.SAS.create_flow_point %{lat: -45.45, lng: 55.45, order: 2, flow_value_id: flow_value4.id}
```

After creating the seeds file, some queries that required a complex relationship between data were tested in the GraphiQL, an interface for making queries using GraphQL that can be installed together with Apollo, the GraphQL library for ReactJS.

The queries selected to be tested were the queries for the Flow Values, the Test Files and TDM Boxes. The Flow Values query was selected because it has a relationship with the Flow Points, Output Files and Span Nodes. The image below shows this query (on the left) and its result (on the right).

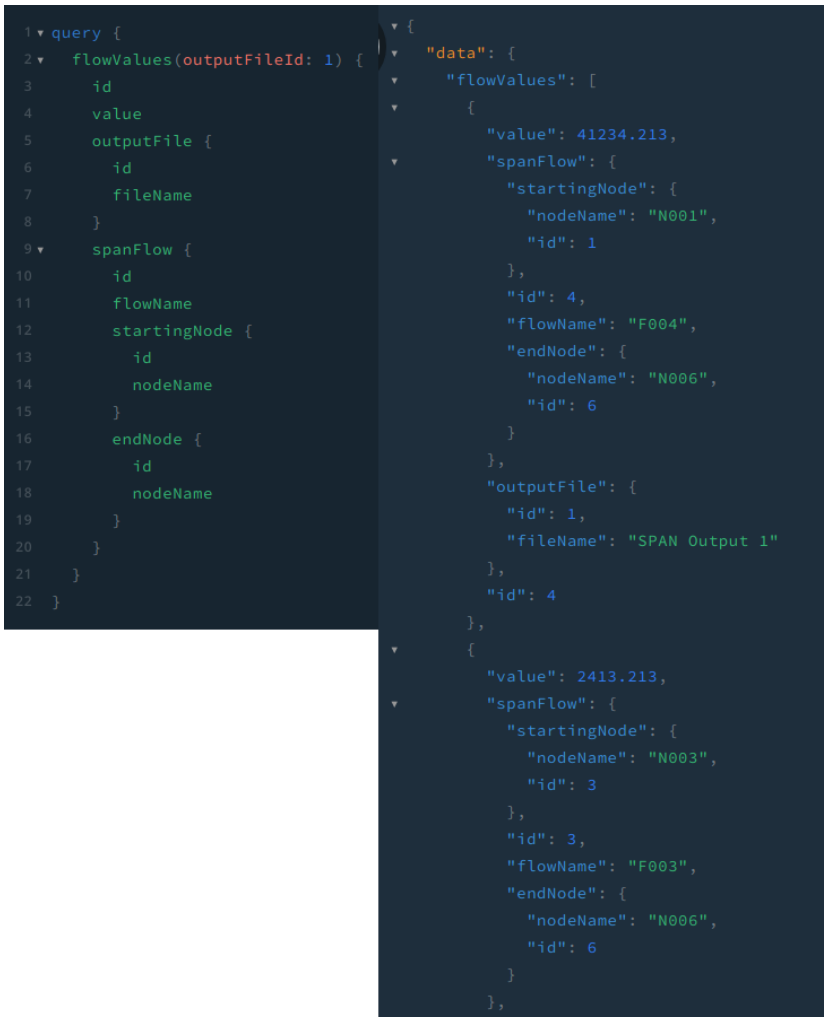


*Figure 24: Flow Values Query*

The next tested query was the query for Test Files, retrieving all Param Values of the file, along with the Test Parameter related to it and the Param Limits related to that Test Parameters. This is presented in the image below.

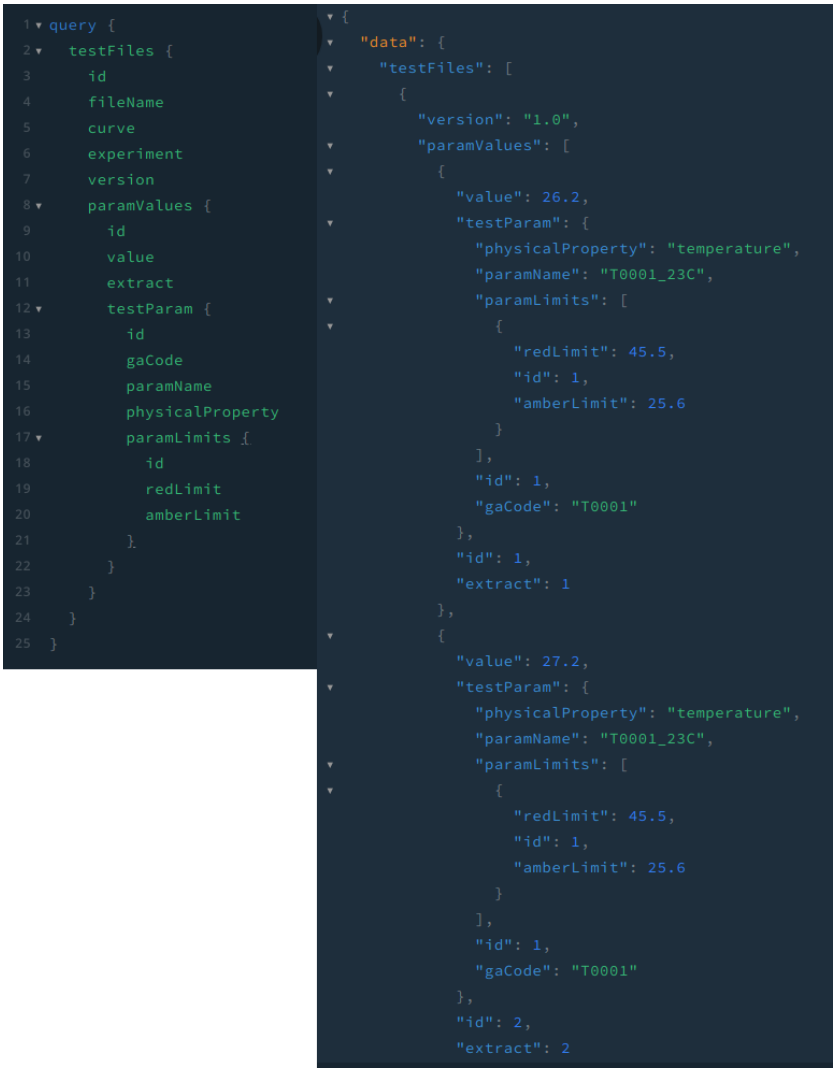


*Figure 25: Test Files Query*

The last tested query was the query for the TDM Boxes, which are the responsible for relating the Test Parameters with the SPAN Nodes. The image below shows this query.

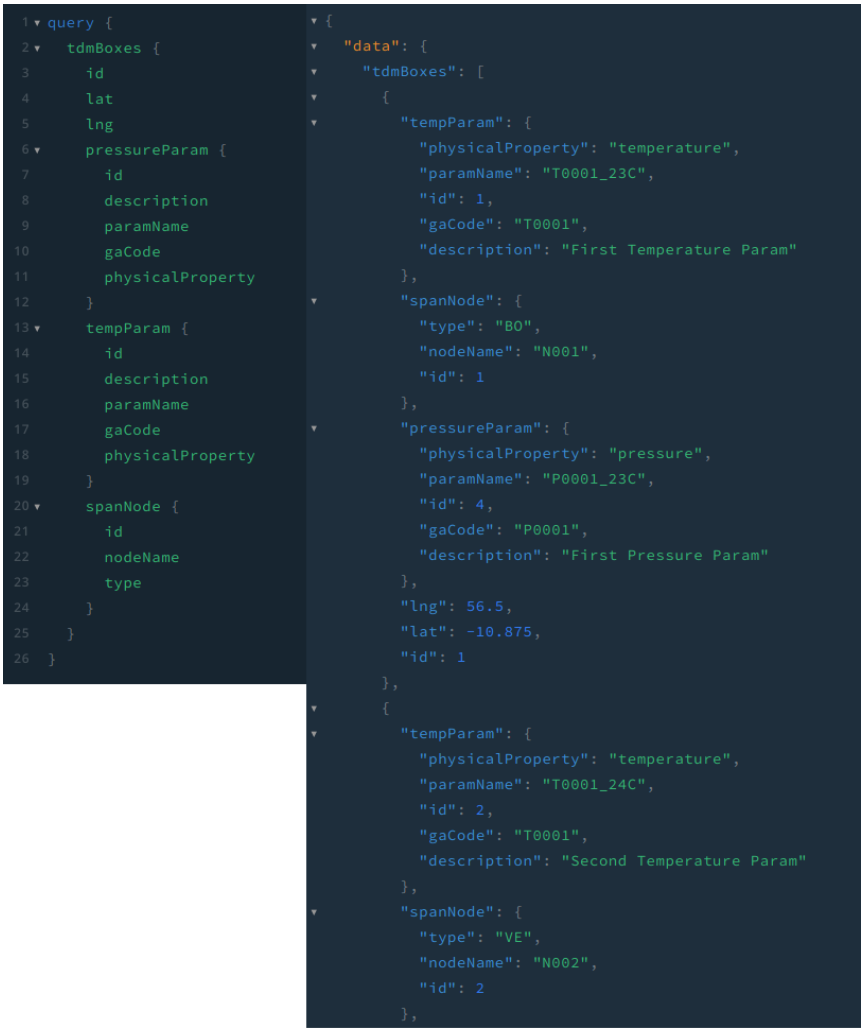


*Figure 26: TDM Boxes Query*

Through the analysis of the result of the query, it was possible to identify that the information was being correctly related and the queries were returning the right data.

### 5.1.2 Functionality and Requirements Check

The next step to check if our application fulfill its requirements was to make a functionality check, comparing it with the requirements table of Appendix A. In a general manner, by simply using it, it was possible to see that the application could compare data from test and simulation, as shown in Figure 23, and visualize simulation data with a large level of details and enabling comparison between them, as it is shown

in Figures 20, 21 and 22, thus achieving the main goal described in section 1.1. However, in order to have a more precise analysis of the results, the list of requirements was reviewed, to see which of the requirements were met.

One thing to consider is that the list of requirements was made for the full version of the application, which covers a lot more than the first version of the application, shown in this document, does. Also, requirements related to user access were dependant of another system that still doesn't classify users in groups, and therefore could not be implemented.

Still, from all the functional requirements, 34 in total, only 7 that were not related to user access were not implemented, and only one not related to user access out of 7 suplementary requirements. The table below list the requirements and presents if they were completely implemented, partially implemented or not implemented at all.

| **Requirement** | **Status** |
|---|---|
| F1 Upload SPAN Files | Implemented completely, apart from user access requirements |
| F2 Upload Test Files | Implemented partially |
| F3 Upload Instrumentation List files | Implemented partially |
| F4 Manage SPAN Files | Implemented completely, apart from user access requirements |
| F5 Manage Test Files | Implemented partially |
| F6 Manage Instrumentation List Files | Implemented partially |
| F7 Edit Instrumentation List Files | Not implemented |
| F8 Save Custom Setup | Not implemented |
| F9 Load Custom Setup | Not implemented |
| F10 Save Current User Setup | Not implemented |
| F11 Load Current User Setup | Not implemented |
| F12 Edit Relationship between Test and SPAN | Implemented completely, apart from user access requirements |
| F13 Edit Scaling and Prediction Data | Implemented completely, apart from user access requirements |
| F14 Select Engine Project | Implemented completely |
| F15 Select SPAN file | Implemented completely |
| F16 Select Engine Build | Implemented completely |

| | |
|---|---|
| F17 Select Test Data | Implemented completely |
| F18 Create New User | Not implemented |
| F19 Edit User | Not implemented |
| F20 Delete User | Not implemented |
| F21 Manage Grouping Tags | Implemented partially |
| F22 Select Plot Parameters | Not implemented |
| F23 Change Measurement Units | Not implemented |
| F24 Change Validation Data | Implemented completely, apart from user access requirements |
| F25 Select Flow Model | Implemented completely |
| F26 Select Master Source | Implemented completely |
| F27 Visualize SPAN Data | Implemented completely |
| F28 Compare SPAN Data | Implemented completely |
| F29 Compare SPAN data with Test Data | Implemented partially |
| F30 Edit Comparison Nodes | Implemented completely, apart from user access requirements |
| F31 Manage Test Only Nodes | Implemented completely, apart from user access requirements |
| F32 Manage Validation Filtering Tags | Not implemented |
| F33 Accuracy Criteria | Implemented partially |
| F34 Manage ERF Association | Not implemented |
| S1 Log of Changes | Implemented partially |
| S2 Tech Stack | Implemented completely |
| S3 Landing Page | Implemented partially |
| S4 URL | Not implemented |
| S5 Close Application | Not implemented |
| S6 Browser Compatibility | Implemented completely |
| S7 Login | Implemented partially |

# 6 CONSIDERATIONS AND PERSPECTIVES

As mentioned in the last item of the previous section, the work presented in this document had as result the complete modelling of an application, with the creation of many documents that describe it and enable the development with an easy understanding of it. Beyond that, an initial version of Engino TDM was implemented, using modern technologies for web-applications. This initial version is already very usefull for improving the process efficiency, and can already be used to reduce the costs of the company.

The cost impact calculated for the full version is of a cost saving ranging from 10.000 euros to 20.000 euros per year. This cost-saving was calculated taking as a base the amount of time that dealing with ineffiencies of the current version takes away per week, which can range from 2.5 to 5 hours for the whole department. Multiplying this value by the number of weeks in a year (54) and by the cost of an engineer for the company (80€ per hour), we get the mentioned value range.

In addition to that, there is a cost saving for each implementation for new engines, which by estimation should take away 20 hours. Multiplying by the cost of an engineer for the company, there is a cost saving of approximately 1.600€ for each new engine added in the TDM.

Beyond that, there are costs that cannot be estimated now, such as the cost saving of integrating the systems in the future, which will have a very high value.

## 6.1 Result Analysis

With the results presented in chapter 5 it was possible to see that the work developed by the student produced a good documentation for the software and also an initial version of it, already working and fullfilling some requirements.

However, the schedule made was done envisioning the complete application. This could not be done, not only because of external tasks that took some time, but also because of the lack of experience in some languages used, something that not only made the implementation much slower, but also caused a wrong prediction of how much time would have to be spent in each task.

For this reason, some parts of the project were left out of the final scope of the project. These parts will be developed by the student itself in the following month, and by the next intern, who will have the complete documentation to help him.

Despite the need to limit the scope of the project, the work developed already produced a very good result, with a good documentation that can serve as guideline for future developers and the initial version of it, which already solves partially the issue presented in the beggining of this document. Also, the complexity of the project was very big, something that adds even more value to the results of this project.

## 6.2 Future Perspectives

The presented work opens a lot of opportunities inside the department for future work. The first step for further development is to deliver the complete version of the application, which would then bring all the benefits mentioned previously in this chapter.

After this first step, the possibility of integration with the systems that provide the data used in Engino TDM should be explored, because this would enable the application to have access to live data, and exclude the need to download the data from one of the sources just to have it uploaded into TDM afterwards.

Beyond this step, there is a possibility to integrate Engino TDM with other systems inside the company, because of the great flexibility it presents. Also, because TDM is now a web-application, there is a lot of room for implementing new functionalities via open-source libraries.

# REFERENCES

WAZLAWICK, R. S. **Análise e Projeto de Sistemas de Informação Orient. a Objetos**, Ed. Campus, 2004.

ELMASRI, R.; NAVATHE S. B. **Sistemas de Banco de Dados.** 6. ed. Editora Pearson, 2011.

HEUSER, C. A. **Projeto de Banco de Dados.** 6. ed. Série Livros Didáticos – Instituto de Informática da UFRGS, número 4. Editora Bookman, 2009.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I.. **The Unified Modeling Language User Guide**. Addison-Wesley, 1999.

Overview – absinthe - HexDocs. Available on: <https://hexdocs.pm/absinthe/>. Accessed on July 02.

Kernel – Elixir – HexDocs. Available on <https://hexdocs.pm/elixir/>. Accessed on July 02.

Phoenix – HexDocs. Available on <https://hexdocs.pm/phoenix/>. Accessed on July 02.

Ecto – HexDocs. Available on <https://hexdocs.pm/ecto/>. Accessed on July 02

ReactJS, React Reference. Available on: <https://reactjs.org/docs/react-api.html>. Accessed on February 19.

Redux – Official Site. Available on: <https://redux.js.org/>. Accessed on July 02.

React Router: Declarative Routing for React.js. Available on: <https://reacttraining.com/react-router/>. Accessed on July 02.

Leaflet – Official Site. Available on: <https://leafletjs.com/>. Accessed on July 02.

GitHub – PaulLeCam/react-leaflet. Available on: <https://github.com/PaulLeCam/react-leaflet>. Accessed on June 29.

Material-UI – Official Site. Available on: <https://material-ui.com/>. Accessed on June 29.

Material Design – Official Site. Available on: <https://material.io/>. Accessed on June 29.

Agile Alliance, The Agile Manifesto. Available on: <https://www.agilealliance.org/agile101/the-agile-manifesto/ >. Accessed on June 24.

GitHub, Facebook, React - Sites Using React. Available on: <https://github.com/facebook/react/wiki/sites-using-react >. Accessed on

June 24.

State of JS. Available on: <http://2016.stateofjs.com/2016/frontend/>. Accessed on June 29.

Carbon Five, Elixir and Phoenix: The Future of Web Apps? Available on: <https://blog.carbonfive.com/2016/04/19/elixir-and-phoenix-the-future-of-webapis-and-apps>. Accessed on June 24.

How to GraphQL, GraphQL is the better Rest. Available on: <https://www.howtographql.com/basics/1-graphql-is-the-better-rest/>. Accessed on June 24.

ReactJS, Thinking in React. Available on: <https://reactjs.org/docs/thinking-in-react.html>. Accessed on May 9.

ReactJS, Components and Props. Available on: <https://reactjs.org/docs/components-and-props.html>. Accessed on May 19.

# APPENDIX A – TABLE OF REQUIREMENTS

Here complete table of requirements for the Engino TDM project is listed.

| F1 Upload SPAN Files | | Hidden ( ) | | |
|---|---|---|---|---|
| **Description:** The system must enable the user to upload the SPAN files, processing them and storing them in the database | | | | |
| ***Non-Function Requirements*** | | | | |
| **Name** | **Restriction** | **Category** | **Mandatory** | **Permanent** |
| NF1.1 Type of data | The data from the SPAN file must be a formatted plain text. If the data is not in the specified format, it should show an error on the interface. | Implement ation | ( X ) | ( ) |
| NF1.2 Access Control | Only people with special user access should be able to upload | Security | ( X ) | ( X ) |
| NF1.3 File types Upload | Two types of files should always be uploaded together, they are the BASE and OUTPUT types. | Integration | ( X ) | ( X ) |

| F2 Upload Test Files | | Hidden ( ) | | |
|---|---|---|---|---|
| **Description:** The system must enable the user to upload the test files, processing them and storing them in the database | | | | |
| ***Non-Function Requirements*** | | | | |
| **Name** | **Restriction** | **Category** | **Mandatory** | **Permanent** |
| NF2.1 Type of data | The data from the text file must be a formatted excel spreadsheet. If the spreadsheet does not contain necessary rows, it should show an error on the interface. | Implement ation | ( X ) | ( ) |
| NF2.2 Access Control | Only people with special user access should be able to upload | Security | ( X ) | ( X ) |
| NF2.3 Default Test Units | The system should have a default set of test units that will be loaded when the user is uploading a file | Interface | ( X ) | ( X ) |
| NF2.4 Test Units | The user should be able to define the units for the file he is uploading. | Interface | ( X ) | ( X ) |

| F3 Upload Instrumentation List files | | Hidden ( ) | | |
|---|---|---|---|---|
| **Description:** The system must enable the user to upload instrumentation list files, processing them and storing them in the database | | | | |
| ***Non-Function Requirements*** | | | | |
| **Name** | **Restriction** | **Category** | **Mandatory** | **Permanent** |
| NF3.1 Type of data | The data from the instrumentation list must be a formatted excel spreadsheet. If the spreadsheet does not contain necessary rows, it should show an error on the interface. | Implement ation | ( X ) | ( ) |

| | | | | |
|---|---|---|---|---|
| NF3.2 Access Control | Only people with special user access should be able to upload | Security | ( X ) | ( X ) |
| NF3.3 Performance Parameters | The user should be required to set the performance parameters when uploading an instrumentation list | Interface | ( X ) | ( X ) |

| F4 Manage SPAN Files | | Hidden ( ) | | |
|---|---|---|---|---|
| **Description:** The system must enable the user to delete SPAN files and edit the information about it | | | | |
| ***Non-Function Requirements*** | | | | |
| **Name** | **Restriction** | **Category** | **Mandatory** | **Permanent** |
| NF4.1 Access Control | Only people with special user access should be able to manage SPAN files | Security | ( X ) | ( X ) |
| NF4.2 Show OUTPUT Files | Only the OUTPUT files should be displayed, not the BASE files | Interface | ( X ) | ( X ) |
| NF4.3 Delete BASE Files | If an OUTPUT file is deleted, the system should check if there is any other OUTPUT file related to the same BASE file. If there isn't, the system should delete the BASE file | Integration | ( X ) | ( X ) |

| F5 Manage Test Files | | Hidden ( ) | | |
|---|---|---|---|---|
| **Description:** The system must enable the user to delete test files and edit the information about it | | | | |
| ***Non-Function Requirements*** | | | | |
| **Name** | **Restriction** | **Category** | **Mandatory** | **Permanent** |
| NF5.1 Access Control | Only people with special user access should be able to manage test files | Security | ( X ) | ( X ) |
| NF5.2 Change Units | The user should be able to change the measurement units for that file | Interface | ( X ) | ( X ) |

| F6 Manage Instrumentation List Files | | Hidden ( ) | | |
|---|---|---|---|---|
| **Description:** The system must enable the user to delete instrumentation list files and edit the information about it | | | | |
| ***Non-Function Requirements*** | | | | |
| **Name** | **Restriction** | **Category** | **Mandatory** | **Permanent** |
| NF6.1 Access Control | Only people with special user access should be able to manage instrumentation list files | Security | ( X ) | ( X ) |
| | | | | |

| F7 Edit Instrumentation List Files | | Hidden ( ) | | |
|---|---|---|---|---|
| **Description:** The system must enable the user to edit the content of instrumentation list files | | | | |
| ***Non-Function Requirements*** | | | | |
| **Name** | **Restriction** | **Category** | **Mandatory** | **Permanent** |
| NF7.1 Access Control | Only people with special user access should be able to edit instrumentation list files | Security | ( X ) | ( X ) |

| | | | | |
|---|---|---|---|---|
| NF7.2 Performance Parameters | The users should have the option to change the performance parameters for the selected instrumentation list | Interface | ( X ) | ( X ) |

| F8 Save Custom Setup | | Hidden ( ) | | |
|---|---|---|---|---|
| **Description:** The system must enable the user to save the current setup of the system as a custom setup. | | | | |
| ***Non-Function Requirements*** | | | | |
| **Name** | **Restriction** | **Category** | **Mandatory** | **Permanent** |
| NF8.1 Stored Variables | The variables that should be saved are:<br>- X plot variables<br>- Y plot variables<br>- Selected Test Data<br>- Selected SPAN Prediction<br>- Selected Extract<br>- Selected Engine Project<br>- Selected Engine Build<br>- Selected Flow Model<br>- Selected Master Source<br>- Selected Nodes<br>- Selected Flow Nodes<br>- Accuracy Criteria<br>- Measurement Units | Interface | ( X ) | ( X ) |
| NF8.2 Access Control | Only people with special user access should be able to save custom setups | Security | ( X ) | ( X ) |

| F9 Load Custom Setup | | Hidden ( ) | | |
|---|---|---|---|---|
| **Description:** The system must enable the user to load custom setups of the system. | | | | |
| ***Non-Function Requirements*** | | | | |
| **Name** | **Restriction** | **Category** | **Mandatory** | **Permanent** |
| NF9.1 Stored Variables | The variables that should be saved are:<br>- X plot variables<br>- Y plot variables<br>- Selected Test Data<br>- Selected SPAN Prediction<br>- Selected Extract<br>- Selected Engine Project<br>- Selected Engine Build<br>- Selected Flow Model<br>- Selected Master Source<br>- Selected Nodes<br>- Selected Flow Nodes<br>- Accuracy Criteria<br>- Measurement Units | Interface | ( X ) | ( X ) |
| | | | | |

| F10 Save Current User Setup | Hidden ( X ) |
|---|---|

| **Description:** The system should automatically save the current setup of the system as the user setup. | | | | |
|---|---|---|---|---|
| ***Non-Function Requirements*** | | | | |
| **Name** | **Restriction** | **Category** | **Mandatory** | **Permanent** |
| NF10.1 Stored Variables | The variables that should be saved are:<br>- X plot variables<br>- Y plot variables<br>- Selected Test Data<br>- Selected SPAN Prediction<br>- Selected Extract<br>- Selected Engine Project<br>- Selected Engine Build<br>- Selected Flow Model<br>- Selected Master Source<br>- Selected Nodes<br>- Selected Flow Nodes<br>- Accuracy Criteria<br>- Measurement Units<br>- Current Screen<br>- Map Center<br>- Map Zoom Level | Interface | ( X ) | ( X ) |
| | | | | |

| F11 Load Current User Setup | | Hidden ( X ) | | |
|---|---|---|---|---|
| **Description:** The system should automatically load it's the saved user setup into the system. | | | | |
| ***Non-Function Requirements*** | | | | |
| **Name** | **Restriction** | **Category** | **Mandatory** | **Permanent** |
| NF11.1 Stored Variables | The variables that should be loaded are:<br>- X plot variables<br>- Y plot variables<br>- Selected Test Data<br>- Selected SPAN Prediction<br>- Selected Extract<br>- Selected Engine Project<br>- Selected Engine Build<br>- Selected Flow Model<br>- Selected Master Source<br>- Selected Nodes<br>- Selected Flow Nodes<br>- Accuracy Criteria<br>- Measurement Units<br>- Current Screen<br>- Map Center<br>- Map Zoom Level | Interface | ( X ) | ( X ) |
| | | | | |

| F12 Edit Relationship between Test and SPAN | Hidden ( ) |
|---|---|
| **Description:** The system must enable the user to change the relationship between SPAN nodes and test parameters | |

| ***Non-Function Requirements*** | | | | |
|---|---|---|---|---|
| **Name** | **Restriction** | **Category** | **Mandatory** | **Permanent** |
| NF12.1 Access Control | Only people with special user access should be able to edit the relationship list | Security | ( X ) | ( X ) |
| | | | | |

| F13 Edit Scaling and Prediction Data | | Hidden ( ) | | |
|---|---|---|---|---|
| **Description:** The system must display the parameters used to calculate the scaling and prediction data for SPAN, and enable users to edit it | | | | |
| ***Non-Function Requirements*** | | | | |
| **Name** | **Restriction** | **Category** | **Mandatory** | **Permanent** |
| NF13.1 Access Control | Only people with special user access should be able to edit the scaling and prediction data | Security | ( X ) | ( X ) |
| | | | | |

| F14 Select Engine Project | | Hidden ( ) | | |
|---|---|---|---|---|
| **Description:** The system must retrieve all the available engine projects and show it to the user, who can then select the desired one | | | | |
| ***Non-Function Requirements*** | | | | |
| **Name** | **Restriction** | **Category** | **Mandatory** | **Permanent** |
| | | | | |

| F15 Select SPAN file | | Hidden ( ) | | |
|---|---|---|---|---|
| **Description:** The system must retrieve all SPAN files for the selected Engine Project and present it to the user, who can then select the desired one | | | | |
| ***Non-Function Requirements*** | | | | |
| **Name** | **Restriction** | **Category** | **Mandatory** | **Permanent** |
| NF15.1 Select Engine Project First | The user must have already selected an Engine project | Interface | ( X ) | ( X ) |
| NF15.2 Select Multiple Sources | In the functionality of comparing SPAN data, the user must be able to select multiple data sources | Interface | ( X ) | ( X ) |

| F16 Select Engine Build | | Hidden ( ) | | |
|---|---|---|---|---|
| **Description:** The system must retrieve all engine builds for the selected Engine Project and present it to the user, who can then select the desired one | | | | |
| ***Non-Function Requirements*** | | | | |
| **Name** | **Restriction** | **Category** | **Mandatory** | **Permanent** |
| NF16.1 Select Engine Project First | The user must have already selected an Engine project | Interface | ( X ) | ( X ) |
| | | | | |

| F17 Select Test Data | Hidden ( ) |
|---|---|

| **Description:** The system must retrieve all test data files for the selected Engine Build and present it to the user, who can then select the desired one | | | | |
|---|---|---|---|---|
| ***Non-Function Requirements*** | | | | |
| **Name** | **Restriction** | **Category** | **Mandatory** | **Permanent** |
| NF17.1 Select Engine Build First | The user must have already selected an Engine Build | Interface | ( X ) | ( X ) |
| | | | | |

| F18 Create New User | | Hidden ( ) | | |
|---|---|---|---|---|
| **Description:** The system must enable super users to create new users | | | | |
| ***Non-Function Requirements*** | | | | |
| **Name** | **Restriction** | **Category** | **Mandatory** | **Permanent** |
| NF18.1 Access Control | Only people with super user access should be able to create a new user | Security | ( X ) | ( X ) |
| NF18.2 Confirmation | The system should ask the user to confirm the action before saving the changes in the database | Interface | ( ) | ( X ) |
| NF18.3 Existing User in LDAP | The system should check if the user ID is in the LDAP database. If not, should throw an error | Security | ( ) | ( ) |

| F19 Edit User | | Hidden ( ) | | |
|---|---|---|---|---|
| **Description:** The system must enable super users to edit the information about other users | | | | |
| ***Non-Function Requirements*** | | | | |
| **Name** | **Restriction** | **Category** | **Mandatory** | **Permanent** |
| NF19.1 Access Control | Only people with super user access should be able to create a new user | Security | ( X ) | ( X ) |
| NF19.2 Confirmation | The system should ask the user to confirm the action before saving the changes in the database | Interface | ( X ) | ( X ) |

| F20 Delete User | | Hidden ( ) | | |
|---|---|---|---|---|
| **Description:** The system must enable super users to delete other users | | | | |
| ***Non-Function Requirements*** | | | | |
| **Name** | **Restriction** | **Category** | **Mandatory** | **Permanent** |
| NF20.1 Access Control | Only people with super user access should be able to delete a user | Security | ( X ) | ( X ) |
| NF20.2 Confirmation | The system should ask the user to confirm the action before saving the changes in the database | Interface | ( X ) | ( X ) |
| NF20.3 Delete Associated Setup | When a user is deleted, the setup associated with his account should also be deleted | Integration | ( X ) | ( X ) |

| F21 Manage Grouping Tags | Hidden ( ) |
|---|---|

| **Description:** The system must enable users to create, edit, and delete tags for parameters, in order to enable the user to plot and compare parameters of different Engine Projects | | | | |
|---|---|---|---|---|
| ***Non-Function Requirements*** | | | | |
| **Name** | **Restriction** | **Category** | **Mandatory** | **Permanent** |
| NF21.1 Access Control | Only people with special user access should be able to create a new user | Security | ( X ) | ( X ) |
| NF21.2 Multiple Parameters | The user must be able to add tags to multiple parameters at once | Interface | ( X ) | ( X ) |
| | | | | |

| F22 Select Plot Parameters | | Hidden ( ) | | |
|---|---|---|---|---|
| **Description:** The user can select different and multiple parameters to be presented on the plot | | | | |
| ***Non-Function Requirements*** | | | | |
| **Name** | **Restriction** | **Category** | **Mandatory** | **Permanent** |
| NF22.1 Select on Click | The user should be able to go to a plot by clicking in a node in the GAs. | Interface | ( X ) | ( X ) |
| NF22.2 Tagging | The system should check for tags of the selected parameters, and retrieve other parameters with the same tag, allowing the user to select if he wants to display these parameters or not | Interface | ( X ) | ( X ) |
| NF22.3 Checkboxes | The plot should display checkboxes so that the user can hide or show parameters | Interface | ( X ) | ( X ) |
| NF22.4 Auto-update | The plot should be updated automatically if the user has changed any information on it, like selecting or unselecting parameters | Interface | ( X ) | ( X ) |
| NF22.5 Scaling | The SPAN data to be shown in the plot should be first pass by the scaling and prediction equations | Interface | ( X ) | ( X ) |

| F23 Change Measurement Units | | Hidden ( ) | | |
|---|---|---|---|---|
| **Description:** The system should allow the user to select the measurement units he wants | | | | |
| ***Non-Function Requirements*** | | | | |
| **Name** | **Restriction** | **Category** | **Mandatory** | **Permanent** |
| NF23.1 Store Conversion Factors | The system should have a database with conversion factors and conversion sums between different measurement units | Interface | ( X ) | ( X ) |
| NF23.2 Auto-Update | The system should be dynamic, so that all the values are recalculated when the measurement units have changed | Interface | ( X ) | ( X ) |
| NF23.3 Parameters | The units should be stored for the following parameters:<br>- Mass Flow<br>- Volume Flow<br>- Pressure<br>- Temperature<br>- Power | Interface | ( X ) | ( X ) |

| | | | | |
|---|---|---|---|---|
| | - Power/Temp<br>- Load<br>- Length<br>- Area<br>- Volume<br>- Angle<br>- Velocity<br>- Rot. Speed | | | |
| NF23.4 Monitor Units Storing | The Monitor units should be associated with the setups | Interface | ( X ) | ( X ) |

| F24 Change Validation Data | | Hidden ( ) | | |
|---|---|---|---|---|
| **Description:** The system should show the parameters and enable the user to toggle which are used to calculate the average for the related points | | | | |
| ***Non-Function Requirements*** | | | | |
| **Name** | **Restriction** | **Category** | **Mandatory** | **Permanent** |
| NF24.1 Auto-Update | The system should be dynamic, so that all the values are recalculated when the one parameter have changed | Interface | ( X ) | ( X ) |
| NF24.2 Colors | The validation data should be shown in different colors, to describe which of the parameter values are according to the accuracy criteria (calculated against the average value) | Interface | ( X ) | ( X ) |
| NF24.3 Toggle values | The user should be able to toggle which parameters are used to calculate the average value | Interface | ( X ) | ( X ) |
| NF24.4 Parameter Grouping | All the parameters should be grouped by the GA code when shown | interface | ( X ) | ( X ) |
| NF24.5 Plots | The user should be able to open a plot coming from this screen. The plot should show In the Y axis the values of each of the composing parameter, and in the X axis the extracts | interface | ( X ) | ( X ) |
| NF24.6 Access Control | Only users with special user access should be able to change validation data | Security | ( X ) | ( X ) |

| F25 Select Flow Model | | Hidden ( ) | | |
|---|---|---|---|---|
| **Description:** The User should be able to select which Flow Model to use in the visualization of nodes and flows on the SPAN comparison functionality | | | | |
| ***Non-Function Requirements*** | | | | |
| **Name** | **Restriction** | **Category** | **Mandatory** | **Permanent** |
| | | | | |

| F26 Select Master Source | Hidden ( ) |
|---|---|

| **Description:** The User should be able to select which Master Source to use in the visualization of nodes and flows on the SPAN comparison functionality | | | | |
|---|---|---|---|---|
| ***Non-Function Requirements*** | | | | |
| **Name** | **Restriction** | **Category** | **Mandatory** | **Permanent** |
| NF26.1 Select Flow Model First | The possibility to select the Master Source should only be available after the user already selected the Flow Model | Interface | ( X ) | ( X ) |
| | | | | |

| F27 Visualize SPAN Data | | Hidden ( ) | | |
|---|---|---|---|---|
| **Description:** The system should enable the user to visualize nodes and flows from the selected master source | | | | |
| ***Non-Function Requirements*** | | | | |
| **Name** | **Restriction** | **Category** | **Mandatory** | **Permanent** |
| NF27.1 Auto-Update | The flows and nodes should be updated automatically if the user has changed the selected master source | Interface | ( X ) | ( X ) |
| NF27.2 Toggle Nodes | The user should be able to toggle the visualization of nodes | Interface | ( X ) | ( X ) |
| NF27.3 Toggle Flow nodes | The user should be able to toggle the visualization of flow nodes | Interface | ( X ) | ( X ) |
| NF27.4 Toggle Flow lines | The user should be able to toggle the visualization of flow lines | Interface | ( X ) | ( X ) |
| NF27.5 Dynamic flow lines | The flow lines should be scaled according to the massflow, and have a higher weight and opacity according to it | Interface | ( X ) | ( X ) |
| NF27.6 Dynamic Visualization | The system should recalculate the weight and opacity of the flows according to the zoom level and view position | Interface | ( ) | ( X ) |
| NF27.7 Scaling | All the data must be scaled with the current settings before being shown | Interface | ( X ) | ( X ) |
| NF27.8 Search Nodes | The user should be able to search for specific nodes or flows, and pan into the node or flow in the case that the node is found | Interface | ( ) | ( X ) |

| F28 Compare SPAN Data | | Hidden ( ) | | |
|---|---|---|---|---|
| **Description:** The system should enable the user to compare data from SPAN | | | | |
| ***Non-Function Requirements*** | | | | |
| **Name** | **Restriction** | **Category** | **Mandatory** | **Permanent** |
| NF28.1 Different Master Sources | The user should be able to compare data from different SPAN master sources | Interface | ( X ) | ( X ) |
| NF28.2 Flows and Nodes | The two different types of data that should be compared are Flows and Nodes | Interface | ( X ) | ( X ) |

| F29 Compare SPAN data with Test Data | | Hidden ( ) | | |
|---|---|---|---|---|
| **Description:** The system should enable the user to compare data from SPAN with data from tests | | | | |
| ***Non-Function Requirements*** | | | | |
| **Name** | **Restriction** | **Category** | **Mandatory** | **Permanent** |
| NF29.1 Validation | The user should be able to see if the validation of that data is among certain limits | Interface | ( X ) | ( X ) |
| NF29.2 Parameters | The system should show both temperature and pressure values | Interface | ( X ) | ( X ) |
| NF29.3 Special Nodes | The view of the comparison between SPAN and test should show the special nodes fixed in the corner of the screen | Interface | ( X ) | ( X ) |
| NF29.4 Toggle Performance Parameters | The user should be able to toggle the visualization of the performance parameters | Interface | ( X ) | ( X ) |
| NF29.5 Filter Nodes | The user should be able to filter the displayed nodes according to the validation filtering tags and according to the grouping tags | Interface | ( X ) | ( X ) |
| NF29.6 Search Nodes | The user should be able to search for specific nodes or flows, and pan into the node or flow in the case that it is found | Interface | ( ) | ( X ) |
| NF29.7 Toggle Validation | The user should have the option to deactivate the display of the validation if he wants | Interface | ( X ) | ( X ) |
| NF29.8 Check ERF | When a user selects a test file to show the information, the system should check if the parameters in that test file are also in the ERF table | Integration | ( X ) | ( X ) |
| NF29.9 Toggle Complete Values | The user should be able to toggle the visualization of only boxes that are complete, that is, with the Span and test data, and with the data present in the ERF table | Interface | ( X ) | ( X ) |

| F30 Edit Comparison Nodes | | Hidden ( ) | | |
|---|---|---|---|---|
| **Description:** The user should be able to add, edit or delete information about test parameters on the comparison nodes | | | | |
| ***Non-Function Requirements*** | | | | |
| **Name** | **Restriction** | **Category** | **Mandatory** | **Permanent** |
| NF30.1 Access Control | Only people with special user access should be able to add, edit or delete information on comparison nodes | Security | ( X ) | ( X ) |
| NF30.2 Verification | When a user save changes, the system should verify if the node information in the database is still equal to the initial information about the node. If it is not, notify the user that some | Interface | ( ) | ( X ) |

| | | | | |
|---|---|---|---|---|
| | change was made and it will be overwritten | | | |
| NF30.3 Values Position | The user should also be able to change the position of the values table | Interface | ( X ) | ( X ) |

| F31 Manage Test Only Nodes | | Hidden ( ) | | |
|---|---|---|---|---|
| **Description:** The user should be able to create, edit, and delete nodes that contain only test parameters information | | | | |
| ***Non-Function Requirements*** | | | | |
| **Name** | **Restriction** | **Category** | **Mandatory** | **Permanent** |
| NF31.1 Access Control | Only people with special user access should be able to manage new nodes | Security | ( X ) | ( X ) |
| NF31.2 Node Position | The user should be able to define the position of test only nodes | Interface | ( X ) | ( X ) |
| NF31.3 Values Position | The user should also be able to change the position of the values table | Interface | ( X ) | ( X ) |

| F32 Manage Validation Filtering Tags | | Hidden ( ) | | |
|---|---|---|---|---|
| **Description:** The user should be able to add tags to parameters, in order to filter them when he is visualizing and changing them | | | | |
| ***Non-Function Requirements*** | | | | |
| **Name** | **Restriction** | **Category** | **Mandatory** | **Permanent** |
| NF32.1 Access Control | Only people with special user access should be able to add filtering tags | Security | ( X ) | ( X ) |

| F33 Accuracy Criteria | | Hidden ( ) | | |
|---|---|---|---|---|
| **Description:** The user should be able to change the accuracy criteria, which are responsible for checking if the pressure and temperature values are inside the defined limit | | | | |
| ***Non-Function Requirements*** | | | | |
| **Name** | **Restriction** | **Category** | **Mandatory** | **Permanent** |
| NF33.1 Setup Configuration | The Accuracy criteria should be stored and loaded along with the setup configurations | Integration | ( X ) | ( X ) |
| NF33.2 Parameters | The parameters that should be defined are:<br>- Pressure accuracy in %<br>- Temperature accuracy in % | Interface | ( X ) | ( X ) |
| NF33.3 Error criteria | The user should be able to define multiple error criteria, such as checking if the absolute value is higher than some limit, if the value is equal some value and if the value contain only certain digits | Usability | ( ) | ( X ) |
| NF33.4 Standard Error Values | If no error criteria is defined, the system should compare with a standard error criteria, which is: The value is lower than -9000 and the only present digit is 9 | Interface | ( X ) | ( X ) |

| F34 Manage ERF Association | | Hidden ( ) | | |
|---|---|---|---|---|
| **Description:** The system must store an ERF table with a relationship between ERF, build number and parameter | | | | |
| ***Non-Function Requirements*** | | | | |
| **Name** | **Restriction** | **Category** | **Mandatory** | **Permanent** |
| NF34.1 Access Control | Only people with special user access should be able to add filtering tags | Security | ( X ) | ( X ) |
| NF34.2 Show Table | The application should show the ERF Association similarly to a table in excel | Interface | ( X ) | ( X ) |
| NF34.3 Editing | The Application should enable the users to edit the association similarly to an excel table | Interface | ( X ) | ( X ) |
| NF34.4 Copying data | The application should enable the user to copy cells from excel files and paste it in the association table in the app | Interface | ( X ) | ( X ) |

| **Supplementary Requirements** | | | | |
|---|---|---|---|---|
| **Name** | **Restriction** | **Category** | **Mandatory** | **Permanent** |
| S1 Log of Changes | The system should store a log of changes made in the database | Implementation | ( ) | ( X ) |
| S2 Tech Stack | In order to be possible to integrate the new application with Engino, the following languages should be used:<br>• Elixir (Phoenix) – Backend<br>• ReactJS – Frontend<br>• GraphQL – Integration<br>• PostgreSQL – Database | Implementation | ( X ) | ( ) |
| S3 Landing Page | The initial page of the application should be the one that is stored in the User Setup. If none is defined, the landing page should be the one of the comparison between SPAN and Test nodes | Interface | ( X ) | ( X ) |
| S4 URL | The system should save the information about the system in the URL. The information saved are:<br>- Engine Project<br>- Map Center<br>- Zoom Level | Interface | ( ) | ( X ) |
| S5 Close Application | When the user closes the application, the system should verify if there aren't any unsaved changes, and notify the user that those changes will be lost. | Interface | ( ) | ( X ) |
| S6 Browser Compatibility | The application should work in Internet Explorer 11 | Compatibility | ( X ) | ( ) |

| | | | | |
|---|---|---|---|---|
| S7 Login | In order for the users to access the application, they must make a login. Only registered users should have access to it. | Security | ( X ) | ( X ) |