



FEDERAL UNIVERSITY OF SANTA CATARINA  
TECHNOLOGY CENTER  
AUTOMATION AND SYSTEMS DEPARTMENT  
UNDERGRADUATE COURSE IN CONTROL AND AUTOMATION ENGINEERING

Isabele Mangini Silva

**Improving Database Management: A Comparative Analysis of Two  
Human-Machine Interface Solutions**

Lyon  
2024

Isabele Mangini Silva

**Improving Database Management: A Comparative Analysis of Two  
Human-Machine Interface Solutions**

Final report of the subject DAS5511 (Course Final Project) as a Concluding Dissertation of the Undergraduate Course in Control and Automation Engineering of the Federal University of Santa Catarina. Orientador: Prof. Leandro Buss Becker, Dr.

Lyon  
2024

### Ficha de identificação da obra

A ficha de identificação é elaborada pelo próprio autor.

Orientações em:

<http://portalbu.ufsc.br/ficha>

Isabele Mangini Silva

**Improving Database Management: A Comparative Analysis of Two  
Human-Machine Interface Solutions**

This dissertation was evaluated in the context of the subject DAS5511 (Course Final Project) and approved in its final form by the Undergraduate Course in Control and Automation Engineering

Florianópolis, June 28, 2024.

Prof. Marcelo De Lellis, Dr.  
Course Coordinator

**Examining Board:**

Prof. Leandro Buss Becker, Dr.  
Advisor  
UFSC/CTC/DAS

Arnaud Genieux, Eng.  
Supervisor  
Company Capgemini Engineering

Bruno Machado Pacheco, Ms.  
Evaluator  
Instituição UFSC/CTC/DAS

Prof. Hector Bessa Silveira, Dr.  
Board President  
UFSC/CTC/DAS

This work is dedicated to my parents for dedicating their  
life supporting all my dreams.

## **ACKNOWLEDGEMENTS**

I would like to thank my parents for always supporting my dreams and providing me with an education that I will carry with me for life. I also want to express my gratitude to the team of engineers at Capgemini Engineering, with whom I completed my PFC during the last six months of my degree. Finally, I extend my deepest thanks to my advisor, Leandro Buss Becker, for his invaluable guidance throughout the development of this dissertation.

## DISCLAIMER

Lyon, February 14th, 2024.

As representative of the Company Capgemini Engineering in which the present work was carried out, I declare this document to be exempt from any confidential or sensitive content regarding intellectual property, that may keep it from being published by the Federal University of Santa Catarina (UFSC) to the general public, including its online availability in the Institutional Repository of the University Library (BU). Furthermore, I attest knowledge of the obligation by the author, as a student of UFSC, to deposit this document in the said Institutional Repository, for being it a Final Program Dissertation ("*Trabalho de Conclusão de Curso*"), in accordance with the *Resolução Normativa n° 126/2019/CUn*.



---

Arnaud Genieux  
Capgemini Engineering

## ABSTRACT

The European market has become increasingly competitive in the creation and development of cutting-edge technology in the area of database management, necessitating that companies rapidly renew and adapt their solutions. Capgemini Engineering, France's leading digital services company and a key player in Europe, must quickly and effectively adapt its products to meet both client and internal needs. This end-of-studies project, a collaboration between Capgemini and the author, aims to modernize an outdated database for one of Capgemini's major clients, in order to align with the latest market solutions for database management. Additionally, the project includes the development of two human-machine interfaces for database operations and version control. This document presents a comparison of the development stack, security, and ease-of-use to better understand the project and the team's solution choices. Key challenges include handling sensitive client data and deploying a solution at a global scale, requiring high levels of compliance. The proposed solution is currently hosted on the company's internal servers and is planned for future deployment.

**Keywords:** Cutting-edge technology. Database. Human-Machine Interfaces.



## RESUMO

O mercado europeu atual tem se mostrado cada vez mais competitivo em criação e desenvolvimento de tecnologia de ponta na área de gerenciamento de banco de dados, fazendo com que empresas precisem renovar e adaptar suas soluções na mesma velocidade. Capgemini Engineering é a principal empresa de serviços digitais da França e uma das líderes europeias, o que a obriga a adaptar seus produtos rapidamente às necessidades tecnológicas dos clientes. É nesse contexto que surge a proposta deste projeto de fim de estudos: em colaboração com a autora deste projeto, a empresa pretende reconstruir uma antiga base de dados de um de seus maiores clientes para uma versão conforme às novas técnicas do mercado. Em paralelo, o projeto também contará com o desenvolvimento de duas interfaces homem-máquina para a realização de operações nessa base de dados e controle de versão. Uma comparação em termos de tecnologia e facilidade de uso será demonstrada neste documento para melhor compreensão do projeto e escolha da equipe em termos de solução. Alguns dos principais desafios encontrados no caminho são questões de tratamento de dados sensíveis do cliente e implantação de uma solução em uma empresa de escala global. A solução proposta encontra-se hoje em servidores internos da empresa para futura implantação.

**Palavras-chave:** Tecnologia de Ponta. Base de Dados. Interface Humano Máquina.

## LIST OF FIGURES

Figure 1 – Realization of Cycle V. . . . .	16
Figure 2 – Environment of the Automatic Pilot. . . . .	18
Figure 3 – Metro Lines for the City of Lyon. . . . .	19
Figure 4 – Train Multiple Unity. . . . .	20
Figure 5 – Use Case Diagram for the CTF Tool Project. . . . .	26
Figure 6 – Sequence Diagram. . . . .	31
Figure 7 – Class Diagram for main Window. . . . .	33
Figure 8 – Class Diagram for Creating a New Test. . . . .	33
Figure 9 – Class Diagram for Deleting a Test. . . . .	34
Figure 10 – Class Diagram for Test View. . . . .	34
Figure 11 – Class Diagram for Test View. . . . .	35
Figure 12 – Schema docker compose. . . . .	38
Figure 13 – venv directory files. . . . .	42
Figure 14 – Client Server Architecture from the developped App. . . . .	44
Figure 15 – Redux architecture. . . . .	45
Figure 16 – REST API Architecture. . . . .	46
Figure 17 – QFrame Class Heritance. . . . .	49
Figure 18 – QWidget Class Heritance. . . . .	50
Figure 19 – VSCode Repertory View. . . . .	52
Figure 20 – Workflow creation sql file. . . . .	53
Figure 21 – UGE et UGTS table schema. . . . .	54
Figure 22 – Differences SQL and NoSQL. . . . .	55
Figure 23 – UI to choose between databases. . . . .	57
Figure 24 – UI search bar. . . . .	57
Figure 25 – UI Add New Test. . . . .	58
Figure 26 – UI Delete Test. . . . .	58
Figure 27 – UI main window HMI Local App. . . . .	59
Figure 28 – UI main window HMI Web Application. . . . .	59

## LIST OF TABLES

Table 1 – Specification of the functional requirement "Modify Test". . . . .	27
Table 2 – Specification of the functional requirement "Select Table". . . . .	27
Table 3 – Specification of the functional requirement "Search Test". . . . .	28
Table 4 – Specification of the functional requirement "Access List of Tests". . .	28
Table 5 – Specification of the functional requirement "Add Test". . . . .	28
Table 6 – Specification of the functional requirement "Remove Test". . . . .	29
Table 7 – Specification of the functional requirement "Export Test". . . . .	30
Table 8 – Specification of the functional requirement "Create Version of Tests".	30
Table 9 – Technologies Overview . . . . .	43
Table 10 – API Endpoints Specification . . . . .	47
Table 11 – Technologies Overview . . . . .	48

## CONTENTS

<b>1</b>	<b>INTRODUCTION</b>	<b>13</b>
1.1	PROJECT OBJECTIVES	13
1.2	PROPOSED SOLUTIONS OVERVIEW	14
1.3	PROJECT SCOPE USING V MODEL METHODOLOGY	15
1.4	OBJECTIVES	16
1.5	DOCUMENT STRUCTURE AND ORGANIZATION	16
<b>2</b>	<b>ENTERPRISE PRESENTATION</b>	<b>18</b>
2.1	GENERAL PRESENTATION	18
2.2	MAGALLY TEAM	18
<b>2.2.1</b>	<b>Automatic Pilot</b>	<b>19</b>
2.2.1.1	Centralized Comand Post	20
2.2.1.2	Automatic Pilot Embedded	20
2.2.1.3	Automatic Pilot Ground	20
<b>3</b>	<b>SOFTWARE REQUIREMENTS SPECIFICATION</b>	<b>22</b>
3.1	GENERAL DESCRIPTION	22
<b>3.1.1</b>	<b>Major Features</b>	<b>22</b>
<b>3.1.2</b>	<b>User Characteristics</b>	<b>23</b>
<b>3.1.3</b>	<b>Assumptions and Dependencies</b>	<b>23</b>
3.2	SYSTEM CHARACTERISTICS AND REQUIREMENTS	24
<b>3.2.1</b>	<b>Functional Requirements</b>	<b>24</b>
3.2.1.1	Use-Case Diagram	24
<b>3.2.2</b>	<b>Specification of Functional Requirements</b>	<b>25</b>
3.2.2.1	Use-Case: Modify Test	25
3.2.2.2	Use-Case: Select Table	26
3.2.2.3	Use-Case: Search Test	27
3.2.2.4	Use-Case: Access List of Tests	27
3.2.2.5	Use-Case: Add Test	28
3.2.2.6	Use-Case: Remove Test	29
3.2.2.7	Use-Case: Export Version of Tests	29
3.2.2.8	Use-Case: Create Version of Tests	29
<b>3.2.3</b>	<b>Sequence Diagram</b>	<b>30</b>
<b>3.2.4</b>	<b>Non-Functional Requirements</b>	<b>32</b>
3.3	CLASS DIAGRAMS	33
<b>4</b>	<b>PROJECT EXECUTION OVERVIEW</b>	<b>36</b>
4.1	DEVELOPMENT ENVIRONMENT	36
<b>4.1.1</b>	<b>Web Application Solution</b>	<b>36</b>
4.1.1.1	IDE	36

4.1.1.2	OS . . . . .	36
4.1.1.3	Container . . . . .	37
4.1.1.4	Docker . . . . .	37
4.1.1.5	Docker Compose . . . . .	37
4.1.1.6	Directory Structure . . . . .	40
<b>4.1.2</b>	<b>Local Application Solution . . . . .</b>	<b>41</b>
4.1.2.1	IDE . . . . .	41
4.1.2.2	OS . . . . .	41
4.1.2.3	Virtualization . . . . .	41
4.2	WEB APPLICATION DEVELOPMENT . . . . .	42
<b>4.2.1</b>	<b>Web Application Backend API . . . . .</b>	<b>42</b>
4.2.1.1	Programming Languages and Frameworks . . . . .	42
4.2.1.2	Client Server Interaction . . . . .	44
4.2.1.3	Redux . . . . .	45
4.2.1.4	REST API . . . . .	46
<b>4.2.2</b>	<b>Local Application Development . . . . .</b>	<b>47</b>
4.2.2.1	Programming Languages and Frameworks . . . . .	47
4.2.2.2	Used PyQt Class Hierarchies . . . . .	49
4.2.2.3	Repertory Organization . . . . .	52
4.3	DATABASE IMPLEMENTATION . . . . .	52
<b>4.3.1</b>	<b>Database Overview . . . . .</b>	<b>53</b>
<b>4.3.2</b>	<b>Relational Database Overview . . . . .</b>	<b>54</b>
4.3.2.1	MySQL database . . . . .	55
4.4	GUI DESIGN . . . . .	57
<b>5</b>	<b>ANALYSIS OF RESULTS . . . . .</b>	<b>60</b>
5.1	CYBER THREATS ANALYSIS . . . . .	60
5.2	ADVANTAGES AND DISADVANTAGES OF EACH SOLUTION . . . . .	60
5.3	OVERVIEW OF PREFERRED SOLUTION . . . . .	62
<b>6</b>	<b>CONCLUSION . . . . .</b>	<b>64</b>
	<b>References . . . . .</b>	<b>65</b>

## 1 INTRODUCTION

A Human Machine Interface (HMI) data management tool was developed as part of a collaborative project with Capgemini Engineering (CAPGEMINI. . . , 2024) for their client Keolys (KEOLYS. . . , 2024), the development of this HMI will be detailed in this document. Keolys, a leading player in the transportation industry, serves as a public service delegate, managing the Lyon public transport network on behalf of SYTRAL Mobility (the organizing authority) (SYTRAL. . . , 2024). Keolys is responsible for the operation of the Lyon public transport network (TCL) (TCL. . . , 2024) as well as the maintenance of its equipment and infrastructure. Every day, their employees ensure the reliability, safety, and comfort of the TCL network.

Despite their longstanding presence in the market, Capgemini Engineering's internal infrastructure for data collection, storage, and handling within the scope of this project has become outdated compared to current market technology standards. For example, the company has not yet implemented a database tool within the team in which this project was developed.

The existing database management involves manual handling of tables in an MS-word document, which is both time-consuming and prone to errors.

Capgemini's mission emphasizes taking full responsibility for technology ownership, architecture design, innovative product development, and overall digital transformation. Their ultimate goal is to validate their clients' products and technologies.

To achieve its objectives in this project, Capgemini Engineering has hired the author of this document to implement the proposed solution, which is divided into three main areas: frontend, backend, and database management. This comprehensive approach aims to modernize the client technical capabilities, ensuring they remain at the forefront of the transportation industry with efficient and up-to-date data management practices.

### 1.1 PROJECT OBJECTIVES

The project arises from the necessity for a technologically advanced database that better meets the needs of the team: a system that replace the current outdated and cumbersome actual method that relies on an MS-word document for performing CRUD (Create, Read, Update, Delete) operations and version tracking. This outdated method makes the task inefficient and prone to human errors. The main objective is to develop a user-friendly interface that facilitates easy and accessible CRUD operations within the test database, while also tracking and recording changes to ensure process traceability and transparency.

The intended users of this application are the Product Assurance (PA) team of the Automatic Pilot (AP) MAGGALY (MAGALLY. . . , 2024) project, who frequently utilize

the CTF (*'Cahier de Test Fonctionnel'*) database, which one consists of a list of tests that insure safety validation in their main project. Validation engineers must update the functional test database during these activities. By transitioning to a robust database management system, the project aims to improve data handling, providing faster and more reliable access. The intuitive interface will enable users to perform CRUD operations without extensive technical knowledge, with features for clear navigation and easy data manipulation. Version control will log all changes, providing a comprehensive history for accountability and auditing. Enhanced accessibility will allow team members to access the database from various locations through a web-based or cloud-enabled platform, supporting multiple users concurrently. Process efficiency will be increased by streamlining workflows and automating routine tasks, reducing manual effort and error potential.

This project aims not only to meet the immediate needs of the AP MAGGALY for a more robust and safe database management, but also to future-proof the system against growing data management demands, providing a solid foundation for ongoing and future testing activities, thereby maintaining high standards of safety validation for the MAGGALY project.

## 1.2 PROPOSED SOLUTIONS OVERVIEW

The proposed solution for this project includes two distinct implementations: one for a local application and one for a web application. For the local application, PyQt (PYQT... , 2024) will be used to create a user-friendly interface that communicates with the backend API (API... , 2024). This choice is driven by PyQt's capabilities in developing robust desktop applications with rich GUI (graphical user interface) features. The backend for this local application will be developed using Python (PYTHON... , 2024), which offers ease of use, extensive libraries for future development, and seamless integration with both PyQt and the MySQL (MYSQL... , 2024) database.

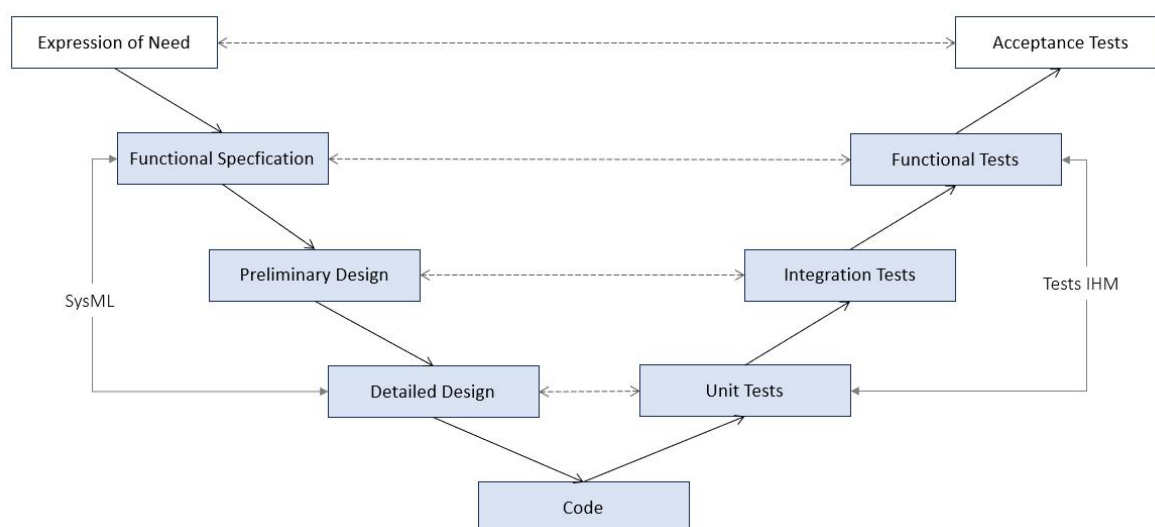
In parallel, a web application will be developed using a modern stack comprising React for the front-end, and Node.js (NODE... , 2024) with Express (EXPRESS... , 2024) for the back-end. This web-based solution aims to provide users with remote access through a browser, offering flexibility and accessibility from any location. The MySQL database will serve both the local and web applications, ensuring consistency and integrity of the data across different platforms. This dual approach ensures that users have the flexibility to choose the application mode that best suits their needs, whether they prefer a robust local application or a convenient web-based interface.

### 1.3 PROJECT SCOPE USING V MODEL METHODOLOGY

V-Model (V..., 2024) was the methodology used for developing this software, also known as the Verification and Validation Model, is a software development methodology that emphasizes the importance of testing at each stage of the development process. It is shaped like the letter “V” to illustrate the relationship between each phase of development and its corresponding phase of testing.



Figure 1 – Realization of Cycle V.



Source: Made by the Author.

By mapping each development phase to a corresponding testing phase, the model helps in identifying and fixing issues early in the development process, thereby improving the overall quality and reliability of the system.

#### 1.4 OBJECTIVES

The primary purpose of this project is to enhance the efficiency and effectiveness of Team MAGALLY in managing the database of functional tests.

For this will be implemented two different HMI applications using two distinct sets of technologies. The objectives are as follows:

- **Design and implement** a robust database management system using SQL to enhance data storage and retrieval processes.
- **Evaluate and prepare** the developed solutions for seamless production deployment, ensuring scalability and reliability.
- **Demonstrate** the capability to export database content into Microsoft Word format, ensuring accessibility and ease of use.

#### 1.5 DOCUMENT STRUCTURE AND ORGANIZATION

This document is structured and organized as follows:

- **Enterprise Presentation:** An overview of the enterprise, Team MAGALLY, and necessary concepts for understanding the project: Chapter 2.

- 
- **Software Requirements Specification:** Functional and non-functional requirements of the project, along with a SysML (SYSML. . . , 2024) analysis of the use cases: Chapter 3.
  - **Project Execution Overview:** Overview of the execution process and development stages for both solutions: Chapter 4.
  - **Analysis of Results:** A review of both solutions, highlighting the positive and negative points: Chapter 5.
  - **Conclusion:** Summarizes the work done, and the objectives achieved: Chapter 6.

## 2 ENTERPRISE PRESENTATION

In this section, the company and details about the project will be presented, clarifying key concepts about the client and the team.

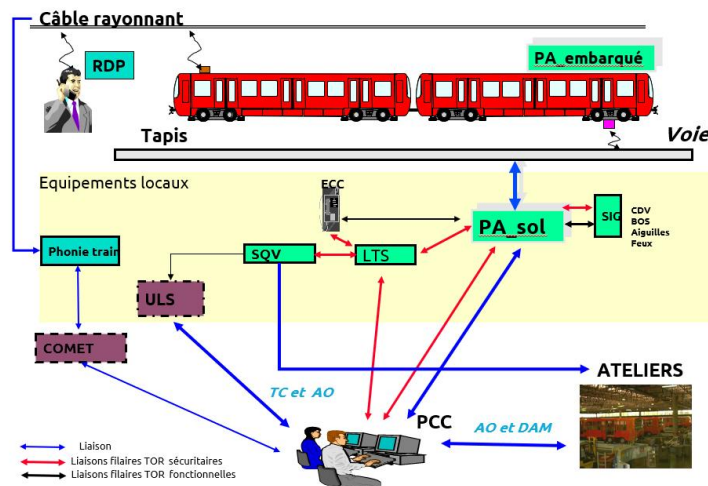
### 2.1 GENERAL PRESENTATION

Capgemini Engineering is one of the world's leading providers of engineering and *R&D* (Research and Development) services. Capgemini Engineering combines its extensive industry knowledge and cutting-edge digital and software technologies to support the convergence of digital worlds. Capgemini Engineering has 60,000 engineers and scientists in more than 30 countries, operating in sectors such as aerospace, space, defense, maritime, automotive, rail, infrastructure and transport, energy, utilities, and IT. Capgemini Engineering Lyon primarily undertakes missions related to factory supervision. These missions involve production management, which includes the monitoring and control of manufacturing systems and data flows at the production workshops of various clients (MES) (MES. . . , 2024). The main objective of an MES is to ensure the effective execution of manufacturing operations and to improve production efficiency.

### 2.2 MAGALLY TEAM

As previously mentioned, the project was developed for the MAGALLY team, specifically for the product assurance team.

Figure 2 – Environment of the Automatic Pilot.



Source: Made by the Author.

The product assurance team is responsible for the maintenance and security requirements of the Lyon Metro Line D. In the Figure 3 it is possible to see in green the line that represents the path of the line D and its stations. Figure 2 illustrates the functioning of the Automatic Pilot of Line D, which is the ensemble of people, machines and embedded systems working together to supervise the behavior of all the metro lines.

Figure 3 – Metro Lines for the City of Lyon.



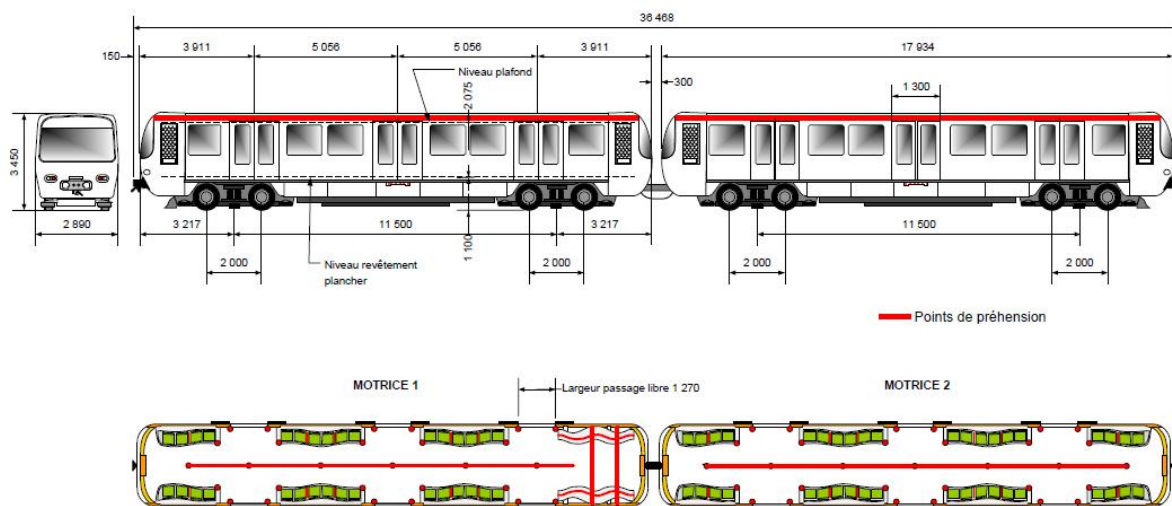
Source: Made by the Author.

### 2.2.1 Automatic Pilot

The AP is an essential system ensuring the operation and safety of trains in single unit (US) or multiple unit (UM) (shown in Figure 4), strictly following the operating instructions transmitted by the Central Command Post. The AP was developed according to several key principles: it enables fully automatic driving of trains without the need for a driver, it manage also the train movement using the deformable moving block principle to ensure a safety zone in front of the train. Safety functions are handled by a coded monoprocessor, and communication between the ground and the train is

entirely digital. The AP equipment includes a redundant on-board management unit (UGE) for each on-board element, and on the ground, a redundant section and section management unit (UGTS), except for the one used on the test track.

Figure 4 – Train Multiple Unity.



Source: Made by the Author.

### 2.2.1.1 Centralized Comand Post

The Central Command Post plays a crucial role in the integrated management of railway operations and passenger services. It ensures the supervision and control of train traffic, passenger reception, station equipment management, and energy network management. It includes a control station dedicated to managing station access.

### 2.2.1.2 Automatic Pilot Embedded

The Embedded Management Unit (UGE) plays a crucial role in the automatic train control system. Each UGE, mounted in a box under the M1 motor car (first single unit train), ensures onboard safety functions and automatic driving. Each train unit includes two redundant UGEs (UGE A and B), each connected to two digital transmission antennas and two reception antennas mounted on the bogie of the M1 motor car. Additionally, each UGE is connected to safety tone wheels mounted on a bogie axle, allowing the measurement of displacement and speed of the unit.

### 2.2.1.3 Automatic Pilot Ground

The Section and Track Management Unit (UGTS) is a ground-based equipment located in the technical rooms of the stations. Each UGTS manages a defined line

area called a material section, divided into automation sections corresponding to the functional breakdown of ground functions. These UGTS ensure ground safety functions, tracking of train units, alarm management, and functional supervision of trains, including interrogation and movement management. Each material section is controlled by two redundant UGTS (UGTS A and UGTS B) interfaced with the PCC to ensure operational continuity. The UGTS are dimensioned considering the maximum transmission distance of ground-to-train messages via the transmission carpet, limited to approximately 1000 meters, and are designed to manage up to two stations per material section.

### 3 SOFTWARE REQUIREMENTS SPECIFICATION

Software requirements specification (SRS) describes the project's functionality, features, design, limitations, and goals. The SRS outlines how the application should operate and how it should be built. The client may use it to define the project expectations and deliverables, the company used it to assess the amount of work, define the technology stack, and estimate the project cost in the beginning of the internship.

#### 3.1 GENERAL DESCRIPTION

The primary purpose of this project is to enhance the efficiency and effectiveness of Team MAGALLY in managing the database of functional tests. This database comprises two principal tables, CTF (catalog of functional tests) UGE and CTF UGTS further explained in this document, which are integral to the company's daily operations and activities. The existing database management involves manual handling of tables in an MS-word document, which is both time-consuming and prone to errors.

The overarching goal of this project is to develop a robust application that can seamlessly handle CRUD operations, thereby automating and streamlining the database management process. Additionally, the project aims to introduce version control mechanisms to the database, allowing for the separation and tracking of different database versions. This will facilitate more comprehensive data analysis and historical tracking of changes over time.

By transitioning from a manual word table to an automated application, the project intends to significantly improve data accuracy, reduce manual workload, and enhance the overall productivity of Team MAGALLY. This document will further elaborate on the functionalities, features, design, and requirements of the proposed application, ensuring it meets the specific needs and expectations of the company.

##### 3.1.1 Major Features

The major features of this project have been discussed above. In this section, the main capabilities of both the local and web interfaces will be introduced:

The database management system stores the following information in a MySQL database:

1. CTF for UGE: A relational database with information on UGE tests;
2. CTF for UGTS: A relational database with information on UGTS tests.

Some other main requirements of the database management system, independent of being local or being Web Application, are such as:

## 1. User-Friendly Interface

- *Intuitive Design*: The interface is designed for ease of use, ensuring a smooth user experience.

## 2. CRUD Operations

- *Create*: Enables adding new records to the database.
- *Read*: Allows viewing and retrieving data efficiently.
- *Update*: Supports modifying existing records.
- *Delete*: Provides options for secure removal of records.

## 3. Version Control

- *Track Changes*: Detailed tracking of modifications over time.
- *Rollback*: Ability to revert to previous versions.
- *Audit Trails*: Comprehensive logs of changes.

## 4. Data Management

- *Search and Filter*: Advanced capabilities to quickly find specific data.
- *Data Validation*: Ensures data meets predefined criteria.
- *Data Import/Export*: Facilitates easy data import and export.

### 3.1.2 User Characteristics

The primary users of this project are the employees of Capgemini Engineering within the MAGALLY Team. This group comprises engineers and engineering interns who will utilize the system in their daily operations. Their technical background and expertise in engineering processes will enable them to effectively leverage the functionalities of the developed application to enhance their workflow and productivity.

### 3.1.3 Assumptions and Dependencies

The dependencies of this software include the operating system of the company laptop used to run the executable file for the local application, which is Windows 11 (WINDOWS. . . , 2021) Professional. For the web application, the dependencies include an internal cloud for database support and deployment, and hosting the website on an internal server with IP address restrictions for security and access management.

For the local application, the database deployment should also be in an embedded database to allow access by multiple users. The executable file (.exe) will be located on a specific shared server.



## 3.2 SYSTEM CHARACTERISTICS AND REQUIREMENTS

For the next sections, the characteristics of the requirements for the software usage and development, as previously defined by the MAGALLY Team Members, will be explored. For the functional requirements, the system will be described according to its behavior under specific conditions, along with user flow definitions and interaction scenarios. Non-functional requirements, on the other hand, are not related to the system's functionality but rather define how the system should perform. They are crucial for ensuring the system's usability, reliability, and efficiency, often influencing the overall user experience.

### 3.2.1 Functional Requirements

The functional requirements outline the specific behaviors and functions that the system must exhibit under various conditions. These requirements are essential for ensuring that the system performs the tasks it is intended to do (PRESSMAN, 2014).

The main functional requirements of this application are:

1. The user is able to **select** the desired table (either UGE or UGTS) from the database through a window after launching the application.
2. It is possible to **search** for a specific test using keywords from the test name column through a search feature.
3. The user can **access** a list of all tests in the database displayed in a table format.
4. Within the test view, users can **modify** the fields of the selected test.
5. A new test can be **added** by clicking a designated "Add Test" button.
6. Any test can be **removed** from the database using a "Delete" button, identified by the test name.
7. Tests can be **exported**, allowing the database to be transferred into an Excel sheet or MS-Word document.
8. The user can **create** different **versions** of the database, categorizing tests according to their respective versions.

#### 3.2.1.1 Use-Case Diagram

As shown in Figure 5, the use case diagram illustrates the various operational capabilities of the CTF Tool Project. Use cases provide detailed information about the system, including the users, the relationships between the system and the users, and the required behavior of the system. In the diagram, actors represent the roles of users

who interact with the modeled system. Accordingly to (COCKBURN, 2000) use cases are crucial for:

- **Bridge Communication Gaps:** Use cases serve as a common language between business stakeholders and technical teams.
- **Drive Development and Testing:** They guide the development of features and creation of test cases, ensuring alignment with user needs.
- **Aid Requirements Clarity:** Use cases help identify functional requirements by capturing real-world scenarios in structured narratives.

### 3.2.2 Specification of Functional Requirements

In this chapter, each functional requirement will be defined in detail. The input refers to the data chosen by the user and applied through the interface. For each case, we have different inputs. The operation describes how the function in the code behaves and interacts with the database. The output is the result displayed to the user in the interface after the backend operation is completed. The priority order for each requirement is defined as follows:

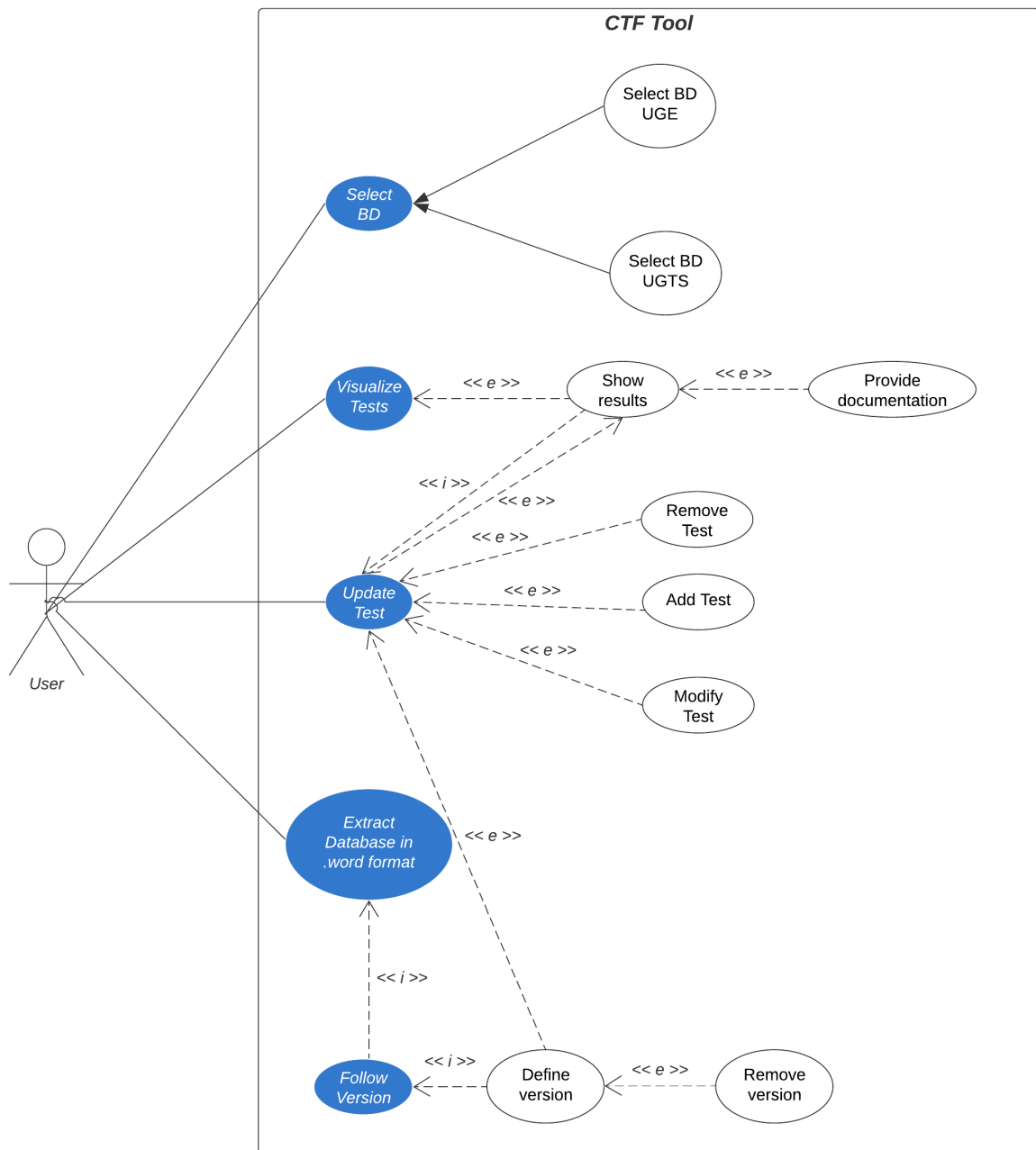
- 1 = High
- 2 = Medium
- 3 = Low

#### 3.2.2.1 Use-Case: Modify Test

##### **Initial description step by step:**

1. The user connects to the IHM.
2. The user has to choose the table he wants to display.
3. The user has the access to all the tests in the database.
4. The user can search for a specific test using words present in its properties.
5. The user modifies one or more properties of the test in a screen designed for this purpose.
6. The test is saved in the database.
7. The modified test is displayed in the first screen.

Figure 5 – Use Case Diagram for the CTF Tool Project.



Source: Made by the Author.

### 3.2.2.2 Use-Case: Select Table

#### Initial description step by step:

1. The user connects to the IHM
2. The first screen allows the user to choose between both tests UGE or UGTS.
3. After choosing, the user has access to all the tests.

Table 1 – Specification of the functional requirement "Modify Test".

Use-Case	Modify Test
<b>Priority</b>	1
<b>Purpose</b>	to apply modifications in a desired test, changing one or more of its properties.
<b>Input</b>	New value of one or more properties.
<b>Operations</b>	The backend function is called, and a SQL request is made in the DB, the database is changed using after an UPDATE is committed in the DB.
<b>Output</b>	Test is updated in the database and shown in the IHM for users review.

Source: Author.

Table 2 – Specification of the functional requirement "Select Table".

Use-Case	Select Table
<b>Priority</b>	1
<b>Purpose</b>	Allow the user to choose one of the two tables.
<b>Input</b>	Table of choice.
<b>Operations</b>	The backend function is called, and a SQL request is made in the DB, the database is changed using after an CREATE TABLE IF NOT EXISTS.
<b>Output</b>	The table is created if already didn't exist.

Source: Author.

### 3.2.2.3 Use-Case: Search Test

#### Initial description step by step:

1. The user connects to the IHM.
2. The user has to choose the table he wants to display.
3. The user has the access to all the tests in the database.
4. The user look for a specific test using words present in its properties.
5. The test searched is displayed in the IHM screen.

### 3.2.2.4 Use-Case: Access List of Tests

#### Initial description step by step:

1. The user connects to the IHM.
2. The user has to choose the desired table to display.
3. The user has the access to all the tests in the database.

Table 3 – Specification of the functional requirement "Search Test".

Use-Case	Search Test
<b>Priority</b>	1
<b>Purpose</b>	Searching for a specific test in the DB using a filter of words.
<b>Input</b>	Word present in one or more properties of the test.
<b>Operations</b>	The backend function is called, and a SQL request is made in the DB, the test is searched using a SELECT and WHERE statements.
<b>Output</b>	Test is found in the table and presented to the user in the IHM.

Source: Author.

Table 4 – Specification of the functional requirement "Access List of Tests".

Use-Case	Access List of Tests
<b>Priority</b>	1
<b>Purpose</b>	Allow the user to have a complete view of all the tests present in the DB.
<b>Input</b>	Previous DB option.
<b>Operations</b>	The backend function is called, and a SQL request is made in the DB, the tests are shown using an SELECT * FROM.
<b>Output</b>	Tests are shown in the IHM allowing the user to scroll from the first to the last.

Source: Author.

### 3.2.2.5 Use-Case: Add Test

#### Initial description step by step:

1. The user connects to the IHM.
2. The user has to choose the table he wants to display.
3. Using the 'Add +' button, the user can insert a new test in the table.

Table 5 – Specification of the functional requirement "Add Test".

Use-Case	Add Test
<b>Priority</b>	1
<b>Purpose</b>	To do the insertion of a new test in the table.
<b>Input</b>	New value for all the properties of the test.
<b>Operations</b>	The backend function is called, and a SQL request is made in the DB, the test is added using INSERT the request is committed and the test is added.
<b>Output</b>	Test is added in the database and shown in the IHM.

Source: Author.

### 3.2.2.6 Use-Case: Remove Test

#### Initial description step by step:

1. The user connects to the IHM.
2. The user has to choose the table he wants to display.
3. Using the 'Delete' button, the user can delete a test if it exists in the table.

Table 6 – Specification of the functional requirement "Remove Test".

Use-Case	Remove Test
<b>Priority</b>	1
<b>Purpose</b>	Delete a test present in the table.
<b>Input</b>	value 'Test' one of the unique properties of the tests.
<b>Operations</b>	The backend function is called, and a SQL request is made in the DB, the test is deleted using a DELETE and WHERE requests in the table.
<b>Output</b>	Test is deleted from the database and removed from the IHM.

Source: Author.

### 3.2.2.7 Use-Case: Export Version of Tests

The use-case Export Version of Tests is supposed to work as the last step for the user to be able to analyze and understand all the modifications applied to the table. This use-case also matches the requirements of the client, which works only with Excel sheets (MICROSOFT. . . , 2024) or Microsoft Word documents.

#### Initial description step by step:

1. The user connects to the IHM.
2. The user has to choose the table he wants to display.
3. Using the Export button in the main screen, the user is able to export all the tests to an Excel file.

### 3.2.2.8 Use-Case: Create Version of Tests

The use-case Create a version of tests is extremely useful to the user, the CTF BD changes with the evolution of the metro D specifications, which means new tests are added, and previous tests are changed to fit the new requirements of the metro D line and new versions are created to better follow these modifications.

Table 7 – Specification of the functional requirement "Export Test".

Use-Case	Export Test
Priority	2
Purpose	Export modified and non modified tests, so the user can analyze each change in a format matching the client's requirements.
Input	Button click.
Operations	The front-end logic deals with the export of the tests by a specific function for it.
Output	A file is created with all the tests and downloaded in the user laptop.

Source: Author.

Table 8 – Specification of the functional requirement "Create Version of Tests".

Use-Case	Create Version of Tests
Priority	2
Purpose	Create a new version of the DB and use it to analyze all the modifications applied in the tests from one version to the other.
Input	All the DB operations and modifications.
Operations	Front-end and Backend functions will be applied to create different versions of the table and store them all in a new table.
Output	Version is created and shown in the IHM.

Source: Author.

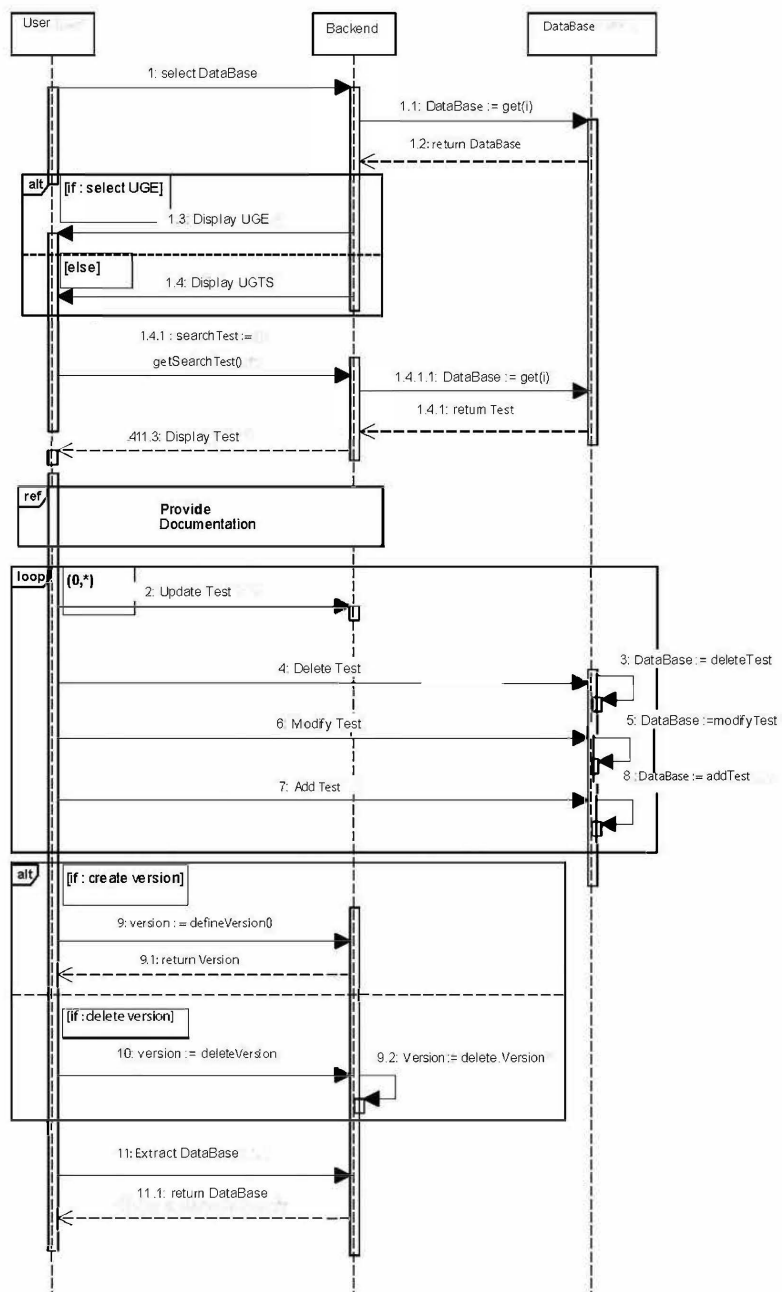
### 3.2.3 Sequence Diagram

Sequence diagrams plays a pivotal role in the development of the project, as it offered a clear visualization of the interactions between different system components. The sequence diagram helped to map out the functional requirements step-by-step. By illustrating the flow of messages between the user, backend services, and database, the sequence diagram ensured that each functional requirement is accurately captured and implemented. This does not only aids in identifying potential issues early in the design phase but also provides a blueprint for following further in the development, ensuring consistency and coherence throughout the project (FOWLER, 2004).

The figure 6 shows the Sequence Diagram for this project. The entities are the user, the Backend, and the Database. For ease of comprehension, the user can make calls to the Backend to accomplish the desired use cases, and the Database provides the necessary data for each user request. Each operation in the Database is accomplished by SQL (SQL. . . , 2024) requests on the Backend side and then shown in an interface for the user.

Figure 6 – Sequence Diagram.

sd [CTF MAGALLY]



Source: Made by the Author.



### 3.2.4 Non-Functional Requirements

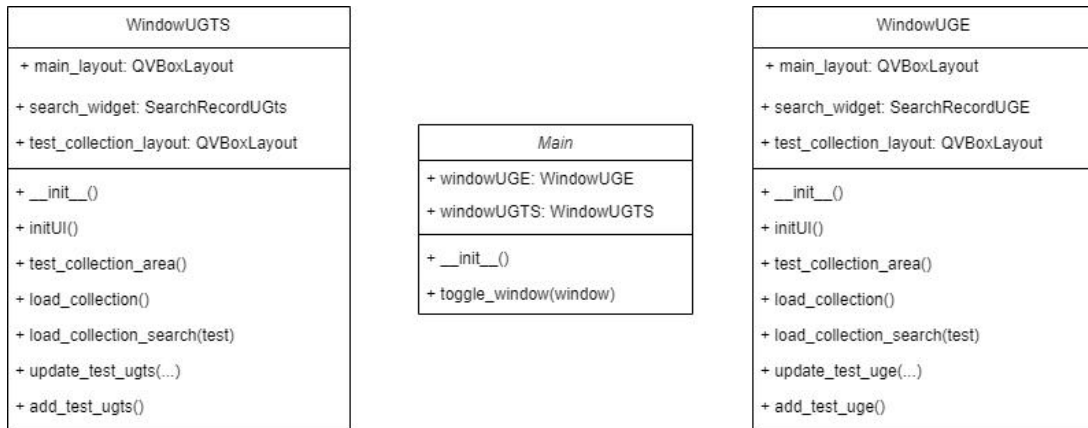
The Non-functional requirements (NFRs) define the quality attributes of the system. While they do not describe specific behaviors, they set critical benchmarks for how the system should perform. NFRs specify criteria such as performance, scalability, reliability, and usability that affect the system's operation it also help ensure that the system adheres to standards and minimizes operational risks (CHUNG et al., 2012).

1. **Compatibility:** The software must be fully compatible with Windows 11, ensuring that all functionalities operate correctly on this OS. It should take advantage of the features and enhancements provided by Windows 11 to deliver an optimal user experience.
2. **Capacity:** The database structure should be robust and scalable, capable of supporting all currently existing tests as well as accommodating future additions. This includes ensuring efficient data storage, retrieval, and management to handle potentially large volumes of data without performance degradation.
3. **Reliability and Availability:** The database should be deployed on a reliable server or Virtual Machine (VM) to ensure high availability. It must be accessible 24/7, minimizing downtime and ensuring that users can access the system at any time of the day. Redundant systems and regular backups should be in place to enhance reliability and data integrity.
4. **Maintainability and Manageability:** The system should be designed with maintainability in mind, allowing future interns or employees to easily manage and improve it. This includes providing comprehensive documentation, modular code design, and clear guidelines for updates and maintenance tasks. The system should support straightforward troubleshooting and debugging processes.
5. **Scalability:** The user interface IHM should be designed to handle concurrent access by multiple users seamlessly. This involves ensuring that the system can scale horizontally or vertically to accommodate increasing numbers of users without compromising performance or user experience. Load balancing techniques may be employed to distribute traffic effectively.
6. **Security:** The interface should be accessible exclusively to Capgemini members, with stringent access controls to ensure that only authorized personnel within the Capgemini internal local network can use it. This includes implementing robust authentication and authorization mechanisms for the Wepp App IHM.

### 3.3 CLASS DIAGRAMS

Figure 7 shows the class diagram for the main window of the application which allows the user to choose between displaying the database UGTS WindowUGTS or UGE WindowUGE.

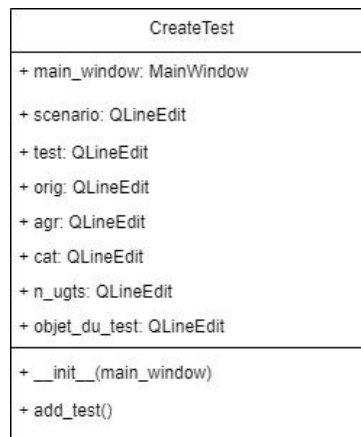
Figure 7 – Class Diagram for main Window.



Source: Made by the Author.

Figure 8 shows the class for creating a new test in the database. It also illustrates how the attributes relate to the information needed for correctly adding a new test. The method `add_test()` uses the information provided by the user to add the test by sending an SQL request to the database with the test attributes.

Figure 8 – Class Diagram for Creating a New Test.

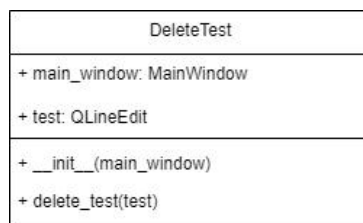


Source: Made by the Author.

Figure 9 demonstrates the class for deleting a test from the database, the user needs to provide only the test name and the method `delete_test()` ensures that the

test is removed from the database.

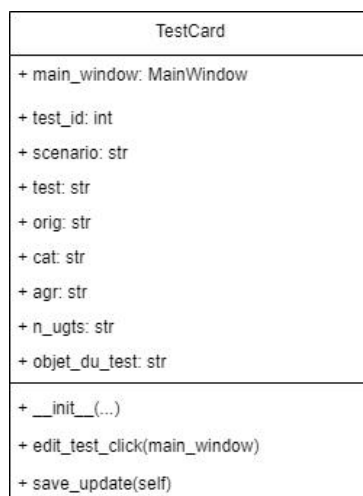
Figure 9 – Class Diagram for Deleting a Test.



Source: Made by the Author.

Figure 10 shows the TestCard Class, which creates the Table View with all the tests and its attributes in a table format. The method `edit_test_click(main_window)` allow the user to edit the test in the grid, and the method `save_update(self)` saves each alteration in the table to the database.

Figure 10 – Class Diagram for Test View.



Source: Made by the Author.

The Figure 11 shows the class SearchTest responsible for finding a test by its test name in the database.

Figure 11 – Class Diagram for Test View.



Source: Made by the Author.

## 4 PROJECT EXECUTION OVERVIEW

In this chapter, the steps and execution of the project will be exposed, as well as the technologies implemented for each solution. Both proposed solutions adhere to the same functional and non-functional requirements outlined earlier. The differences lie in the frameworks and programming languages used to implement each one.

To differentiate between the solutions, we will use the terms “Local Application” for the application developed in Python, and “Web Application” for the web application developed in Node.js. The Local Application operates as an executable file on a computer, built with PyQt, and does not require an internet connection for launching the system. In contrast, the Web Application, developed with React for the frontend and Node.js for the backend, requires an internet connection to perform database operations, enabling remote access through a web browser.

### 4.1 DEVELOPMENT ENVIRONMENT

The development environment encompasses the tools, software, and configurations used to create and test the project. In the scope of this project, the development environment includes: integrated development environment (IDE), programming languages, frameworks and libraries, database, version control and operating system (OS).

#### 4.1.1 Web Application Solution

##### 4.1.1.1 IDE

For this project, it was used the IDE Visual Studio Code (VISUAL. . . , 2024): Visual Studio Code is a versatile and powerful IDE that supports a wide range of programming languages and extensions. It offers a user-friendly interface, integrated terminal, and robust debugging capabilities. The extensive plugin ecosystem allows for enhanced functionality, such as linting, code formatting, and version control integration with GitLab (GITLAB. . . , 2024), used in this project for versioning of the code.

##### 4.1.1.2 OS

Linux (LINUX. . . , 2024) and Ubuntu (UBUNTU. . . , 2024) were the preferred operating system and distribution for web application development due to stability, security, and performance. It is widely used in server environments, making it easier to develop, test, and deploy applications in a consistent environment. Additionally, Linux supports a vast array of development tools and frameworks, which are essential for efficient web development (NEGUS, 2014).

#### 4.1.1.3 Container

Containers were used alongside other tools in this project due to its many advantages. One of them being the ability to streamline development and deployment processes (SARISHMA, 2021). Containers, particularly through technologies like Docker, have become an essential part of modern software development, offering benefits such as portability, efficiency, and scalability. A container encapsulates an application along with all its dependencies, including libraries, configurations, and runtime environments.

#### 4.1.1.4 Docker

Docker has been extensively studied for its impact on software development and deployment due to its ability to containerize applications.

Docker (DOCKER. . . , 2024) is a platform that enables developers to automate the deployment of applications inside lightweight, portable containers. These containers encapsulate an application and its dependencies, allowing it to run consistently across any environment, whether on a developer's machine, a test server, or in production. Containers are isolated from each other and from the host system, providing a secure and efficient environment for running applications (MERKEL, 2014).

Docker containers (CONTAINERS. . . , 2024) are created from Docker images, which are read-only templates that include the instructions for creating a container. These images can include the application code, runtime, libraries, and other dependencies necessary for the application to run.

#### 4.1.1.5 Docker Compose

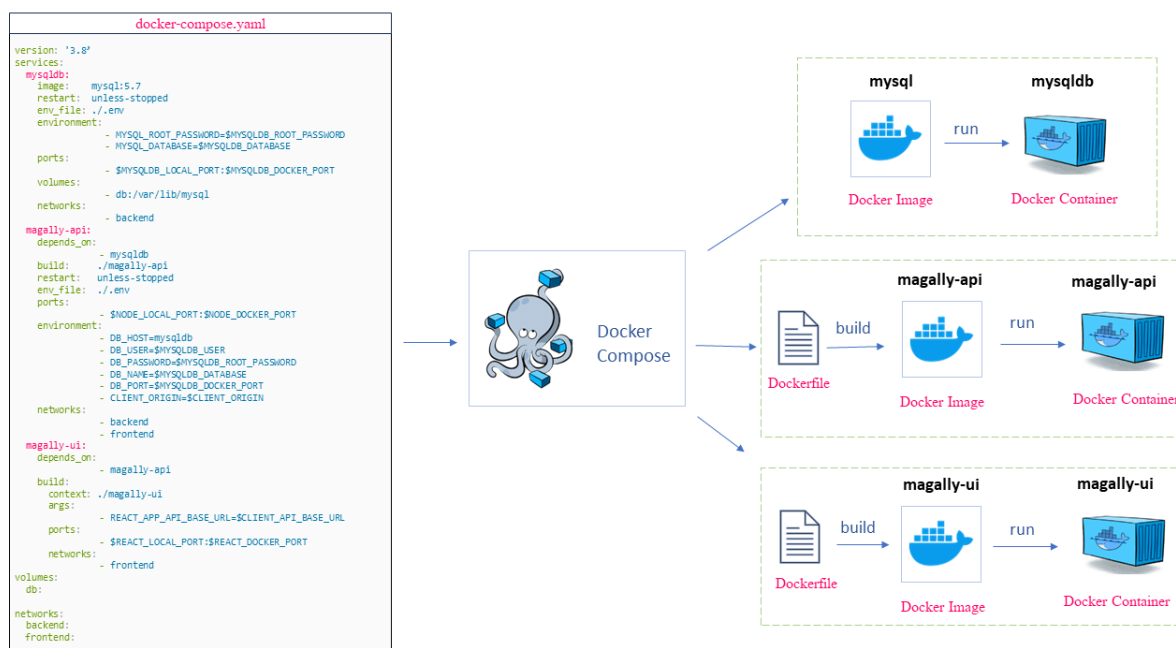
Docker Compose is a powerful tool that simplifies the management of multi-container applications. By defining application services, networks, and volumes in a single configuration file. For instance, Docker Compose is particularly effective in microservices-based architectures, where each service (e.g., frontend, backend, and database) runs in its own container .

The use of Docker Compose (DOCKER. . . , 2024) for this project offered two main advantages:

- **Efficiency:** Unlike traditional virtual machines, Docker containers share the host operating system's kernel, which reduces overhead and improves performance;
- **Isolation:** Containers run in isolated environments, providing security and preventing conflicts between applications.

The figure show the schema for the use of docker compose in this application.

Figure 12 – Schema docker compose.



Source: Made by the Author.

The Docker Compose file defines a multi-container Docker application with three services: `mysql`, `magally-api`, and `magally-ui`. It uses version 3.8 of the Docker Compose specification.

- `mysql` service:
  - Uses the MySQL 5.7 Docker image.
  - Restart policy: The container will restart unless stopped.
  - Environment variables:
    - \* Reads from the `.env` file.
    - \* Sets the MySQL root password and database name.
  - Ports:
    - \* Maps the host port defined by `$MYSQLDB_LOCAL_PORT` to the container port defined by `$MYSQLDB_DOCKER_PORT`.
  - Volumes:
    - \* Uses a named volume `db` to persist MySQL data.
  - Networks:
    - \* Connects to the `backend` network.
- `magally-api` service:

- Depends on:
  - \* Starts after the `mysqldb` service.
- Build:
  - \* Builds the API application from the `./magally-api` directory.
- Restart policy: The container will restart unless stopped.
- Environment variables:
  - \* Reads from the `.env` file.
  - \* Configures the database connection and client origin.
- Ports:
  - \* Maps the host port defined by `$NODE_LOCAL_PORT` to the container port defined by `$NODE_DOCKER_PORT`.
- Networks:
  - \* Connects to both the `backend` and `frontend` networks.
- `magally-ui` service:
  - Depends on:
    - \* Starts after the `magally-api` service.
  - Build:
    - \* Builds the UI application from the `./magally-ui` directory.
    - \* Uses `$CLIENT_API_BASE_URL` as a build argument for the API base URL.
  - Ports:
    - \* Maps the host port defined by `$REACT_LOCAL_PORT` to the container port defined by `$REACT_DOCKER_PORT`.
  - Networks:
    - \* Connects to the `frontend` network.
- Volumes:
  - `db`: A named volume for persisting MySQL data.
- Networks:
  - `backend`: For communication between `mysqldb` and `magally-api`.
  - `frontend`: For communication between `magally-api` and `magally-ui`.



#### 4.1.1.6 Directory Structure

This is the root directory of the project, which contains two main subdirectories `magally-api` and `magally-ui` and the `docker-compose.yml` file: separating the backend (`magally-api`) and frontend (`magally-ui`) allows for modular development. Each component can be developed, tested, and deployed independently. It enables scaling each part of the application independently based on its load. For instance, the backend can be scaled up without affecting the frontend. Each component has its own `Dockerfile`, allowing it to be containerized and deployed consistently across different environments. This isolation helps in managing dependencies and configurations specific to each component.

```
magally-APP
├── magally-api
│   ├── app/
│   ├── .env.sample
│   ├── Dockerfile
│   ├── package.json
│   └── server.js
├── text1.2
│   ├── public/
│   ├── src/
│   ├── Dockerfile
│   ├── package.json
│   └── .env
└── docker-compose.yml
```

- **magally-api/**

- This directory contains the backend (API) part of the application built using Node.js.
  - \* `app/`: This includes the application-specific code, such as controllers, models, routes, etc.
  - \* `Dockerfile`: Defines the Docker image for the backend. It includes instructions for building the Node.js application, such as copying the code, installing dependencies, and setting up the execution command.

- \* `server.js`: The entry point of the Node.js application, where the server is set up and started.

- **magally-ui/**

- This directory contains the frontend part of the application built using React.

- \* `public/`: Contains static files that are served by the web server, such as the `index.html` file.
- \* `src/`: Contains the source code for the React application, including components, utilities, and styles.
- \* `Dockerfile`: Defines the Docker image for the frontend. It includes instructions for building the React application and serving it using a web server like Nginx.

## 4.1.2 Local Application Solution

### 4.1.2.1 IDE

For the local application development, Visual Studio Code remains an ideal choice due to its flexibility and support for multiple languages and frameworks. Its powerful extensions for Python development, such as IntelliSense and Pylint in the tests phase, enhance coding efficiency and maintain high code quality (TEAM, 2023).

### 4.1.2.2 OS

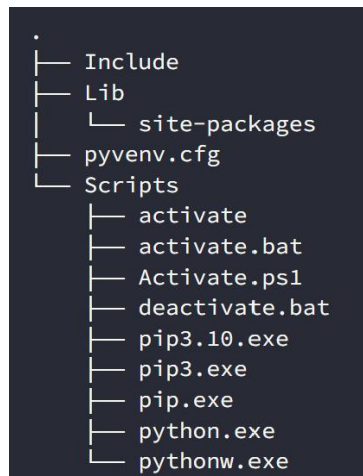
Windows is chosen for the local application development to ensure compatibility with a wide range of hardware and software configurations used to the employees of the company. It provides support to various development tools and frameworks necessary for building robust local applications that use Python.

### 4.1.2.3 Virtualization

Python venv (PIPENV..., 2024) is a built-in module that creates isolated Python environments, allowing the management of dependencies for the project separately. This isolation ensured that the local application's dependencies do not conflict with other projects, leading to more stable and predictable development and deployment processes. With Python venv it is possible to maintain a clean and organized development environment for Python applications.

In summary, these files and directories are designed to ensure that the virtual environment can operate independently of the system Python installation, allowing for different projects to have different dependencies and versions without conflict.

Figure 13 – venv directory files.



Source: Made by the Author.

## 4.2 WEB APPLICATION DEVELOPMENT

This section highlights the Web Application and delves into its development process.

### 4.2.1 Web Application Backend API

The backend of a web application refers to the server-side components that power the application's functionality and manage data processing, storage, and communication. It forms the backbone of the application, handling tasks such as database interactions, user authentication, business logic execution, and integration with external services or APIs. It also ensures the secure, efficient, and scalable operation of web applications by managing data flow and enabling dynamic interactions. It is integral to delivering personalized content, maintaining data integrity, and supporting real-time communication features (FOWLER, 2002).

The explanation of the Web App will be separated into REST API (REST..., 2024) explanation and the frameworks used for implementing all the functional requirements.

#### 4.2.1.1 Programming Languages and Frameworks

As shown in Table 9, the development environment includes various sets of frameworks that serve different purposes. Node.js provides a runtime environment, while React.js (REACT..., 2024) and Redux (REDUX..., 2024) are used for building and managing the user interface. Express is utilized for backend development, and Axios (AXIOS..., 2024) simplifies HTTP (HTTP..., 2024) requests.

Table 9 – Technologies Overview

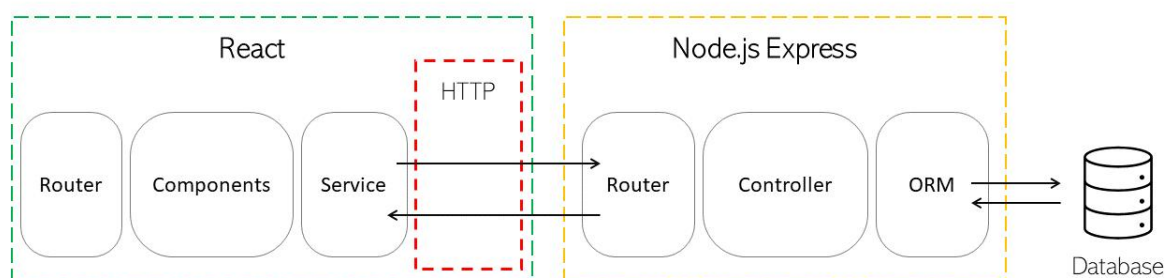
Technologie	Usage	Description
<b>Node.js</b>	Runtime Environment	Node.js is a runtime environment that allows JavaScript (JAVASCRIPT. . . , 2024) to be executed on the server-side. It is built on Chrome's V8 JavaScript engine and is used to create scalable network applications.
<b>React.js</b>	Library	React.js is a JavaScript library for building user interfaces, particularly single-page applications where data can change over time without requiring a page reload. It is maintained by Facebook and a community of individual developers and companies.
<b>Express</b>	Framework	Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications. It is used for building the backend of web applications and APIs.
<b>Axios</b>	Library	Axios is a promise-based HTTP client for the browser and Node.js. It makes it easier to send asynchronous HTTP requests to REST endpoints and perform CRUD operations.
<b>Redux</b>	Library	Redux is a predictable state container for JavaScript applications. It helps manage the state of an application in a predictable way, making it easier to debug and test.
<b>Sequelize</b>	ORM	Sequelize (SEQUELIZE. . . , 2024) is a promise-based Node.js Object-Relational Mapper (ORM) for PostgreSQL, MySQL, MariaDB, SQLite, and Microsoft SQL Server. It features solid transaction support, relations, eager and lazy loading, read replication and more.

Source: Author.

#### 4.2.1.2 Client Server Interaction

The application was built with the following architecture:

Figure 14 – Client Server Architecture from the developed App.



Source: Made by the Author.

- **Node.js Express:**

- Exports REST APIs for handling various HTTP requests such as GET, POST, PUT, and DELETE.
- Interacts with the MySQL database using **Sequelize ORM**, which provides a promise-based interface for database operations, ensuring smooth and efficient data handling.

- **React Client:**

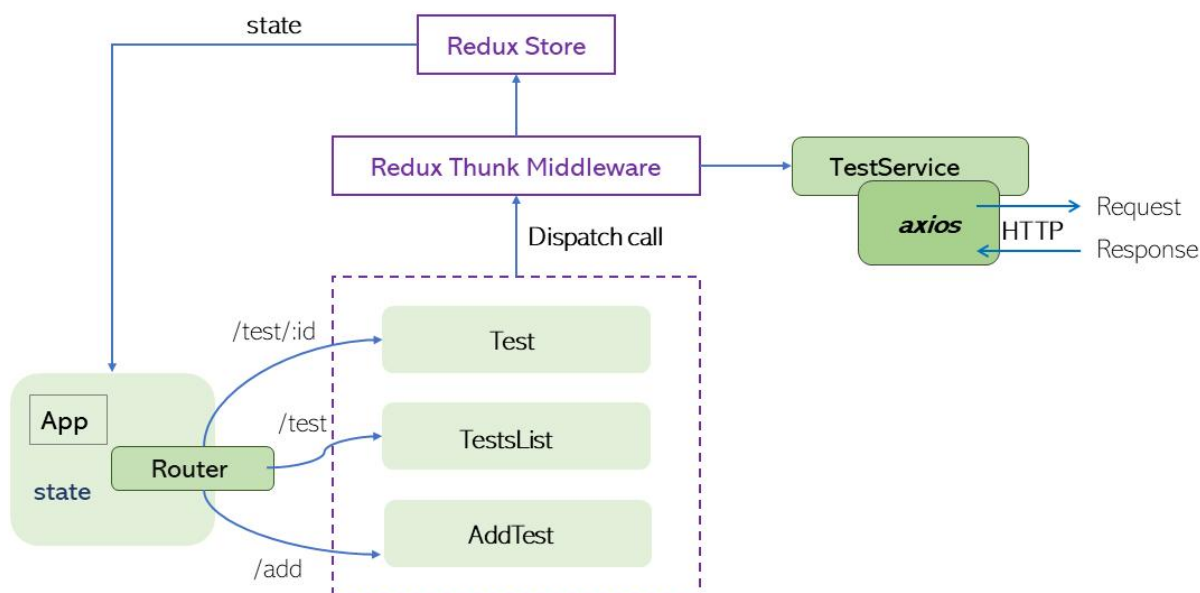
- Sends HTTP requests to the Node.js Express backend and retrieves HTTP responses using **Axios**, a promise-based HTTP client that simplifies the process of performing asynchronous HTTP requests.
- Consumes data retrieved from the backend within the React components, ensuring dynamic and responsive user interfaces.
- Utilizes **React Router** for navigation, enabling seamless transitions between different pages and components within the application.

This architecture ensures a clear separation of concerns, with Node.js handling the server-side logic and database interactions, while React manages the client-side user interface and interactions.

### 4.2.1.3 Redux

The figure 18 shows the architecture of the component Redux used in this project and its relation with the API endpoints and Front-End pages:

Figure 15 – Redux architecture.



Source: Made by the Author.

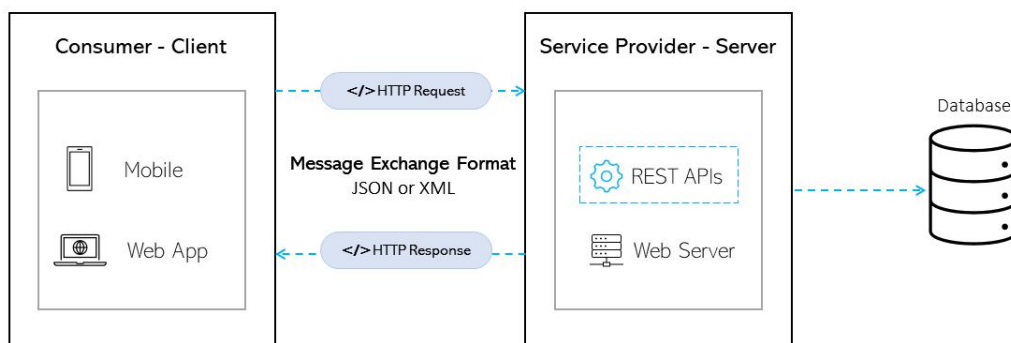
- The **App** component is a container with React Router. It has a navigation bar that links to various route paths.
- There are three main components that interact with Redux Thunk Middleware to call the REST API via the **TestService** which uses **axios** for HTTP requests and responses.
  - **TestsList**: This component retrieves and displays a list of tests.
  - **Test**: This component includes a form for editing a test's details based on its `:id`.
  - **AddTest**: This component includes a form for submitting a new test.
- **Redux Store**: Stores (REDUX. . . , 2024) the application's state.
- **Redux Thunk Middleware**: Allows for writing action creators that return a function instead of an action, enabling asynchronous operations.
  - When a component dispatches an action, Redux Thunk Middleware intercepts the dispatch to perform asynchronous operations before the action reaches the Redux Store.

- **TestService:** Uses **axios** to make HTTP requests to the REST API and handle the responses.
  - **axios:** A promise-based HTTP client for the browser and Node.js used for making asynchronous HTTP requests to the REST API.

#### 4.2.1.4 REST API

A REST API (Representational State Transfer Application Programming Interface) as shown in the figure 18 is a set of rules and conventions for building and interacting with web services. It uses standard HTTP methods and follows the principles of REST architecture to facilitate communication between clients and servers.

Figure 16 – REST API Architecture.



Source: Made by the Author.

The core principles of REST, outlining its key characteristics and the mechanisms it employs to facilitate efficient and scalable web services are:

- **Stateless:** Each request from a client to a server must contain all the information needed to understand and process the request. The server does not store any client context between requests.
- **Client-Server Architecture:** The client and server are independent of each other. The client only knows the URI of the requested resource and acts independently from the server.
- **Uniform Interface:** REST APIs use a uniform interface, typically achieved through standard HTTP methods such as:
  - **GET:** Retrieve data from the server.

- **POST**: Submit data to be processed to the server.
  - **PUT**: Update existing data on the server.
  - **DELETE**: Remove data from the server.
- **Resources and URIs**: Resources are identified by URIs. For example, `/api/books` represents a collection of books, and `/api/books/1` represents a specific book with an ID of 1.

The table 10 show all the Endpoints used for this project and the actions done by the client as a request to the server:

Table 10 – API Endpoints Specification

Methods	Urls	Actions
<b>GET</b>	<code>api/tests</code>	get all Tests
<b>GET</b>	<code>api/tests/:id</code>	get Tests by id
<b>POST</b>	<code>api/tests</code>	add new Test
<b>PUT</b>	<code>api/tests/:id</code>	update Test by id
<b>DELETE</b>	<code>api/tests/:id</code>	remove Test by id
<b>DELETE</b>	<code>api/tests</code>	remove all Tests
<b>GET</b>	<code>api/tests?test={kw}</code>	find all Tests which test title contains 'kw'

Source: Author.

## 4.2.2 Local Application Development

### 4.2.2.1 Programming Languages and Frameworks

For the development of the Local App, the following tools and frameworks were chosen, as shown in Table 11. Overall, Python, PyQt, MySQL, and QtDesign (QT..., 2024) complement each other to create a robust, cross-platform application with sophisticated GUI and database functionality. Python served as the glue that integrated PyQt, MySQL, and QtDesign into a cohesive Local Application.



Table 11 – Technologies Overview

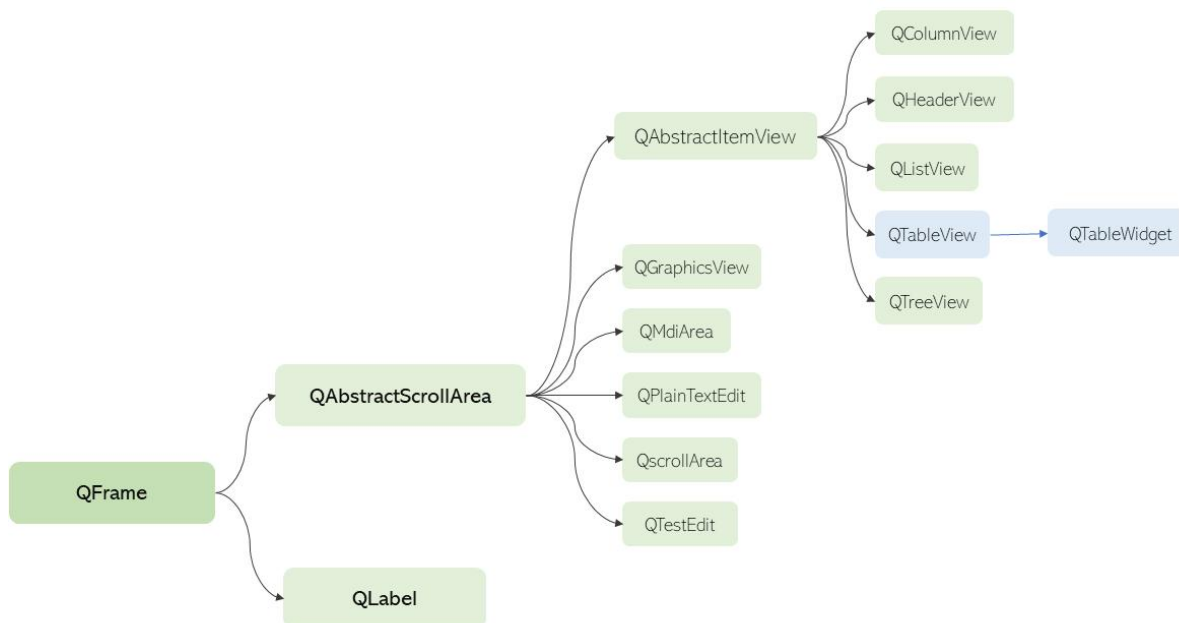
<b>Technology</b>	<b>Usage</b>	<b>Description</b>
<b>Python</b>	Programming Language	Python is a high-level programming language known for its simplicity and readability. It is widely used for web development, data analysis, artificial intelligence, and more.
<b>Python PyQt</b>	GUI Toolkit	Python PyQt is a set of Python bindings for the Qt application framework developed by Riverbank Computing. It allows Python programmers to utilize the powerful Qt libraries to create cross-platform applications with a native look and feel.
<b>MySQL</b>	Database	MySQL is an open-source relational database management system (RDBMS) known for its reliability, scalability, and ease of use. It is commonly used for web applications, data warehousing, and e-commerce.
<b>QtDesign</b>	UI Design Tool	QtDesign, also known as Qt Designer, is a graphical user interface design tool included with Qt that enables developers to design UIs using a drag-and-drop interface. It generates XML files that can be integrated into PyQt applications, making UI design more efficient and intuitive.

Source: Author.

#### 4.2.2.2 Used PyQt Class Hierarchies

For this project the following class heritancy can be observed during the implementation:

Figure 17 – QFrame Class Heritance.



Source: Made by the Author.

- **QWidget (Widget):**

- Represents a rectangular region on the screen.
- Can include buttons, text boxes, labels, etc.
- Can be standalone windows or embedded within other widgets.

- **QDialog (Dialog):**

- Subclass of QWidget that provides a dialog window.
- Used for prompting the user for input or displaying information.
- Can be modal (blocking) or modeless (non-blocking).

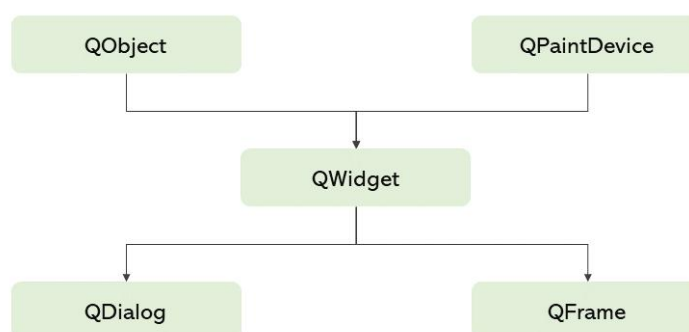
- **QFrame (Frame):**

- Subclass of QWidget that provides a container for other widgets.
- Used for grouping related widgets together or providing visual separation.
- Can have different shapes, styles, and border types.

- **QObject:**

- Base class for all Qt objects.
- Provides support for signals and slots for inter-object communication.
- Provides features such as object name, parent-child relationships, and dynamic properties.

Figure 18 – QWidget Class Heritage.



Source: Made by the Author.

For this project one of its main features is the table displayed for the user in one of its QFrame, a detailed explanation is done to differentiate the usage of QTableView and QTableWidgetItem:

- **QTableView:**

- Part of the model/view framework, displays data from a model.
- Does not store data itself; requests data from a model conforming to QAbstractItemModel.
- Allows for flexible and dynamic data handling with custom models and delegates.
- Ideal for applications with dynamic data, such as database-driven apps.

- **QTableWidget:**

- Convenience class that uses an item-based data model.
- Manages its own data using QTableWidgetItem objects.
- Provides a simpler and more user-friendly API for table management.
- Suitable for simpler applications with static or easily managed data, like spreadsheets or settings tables.

**Example in a code:**

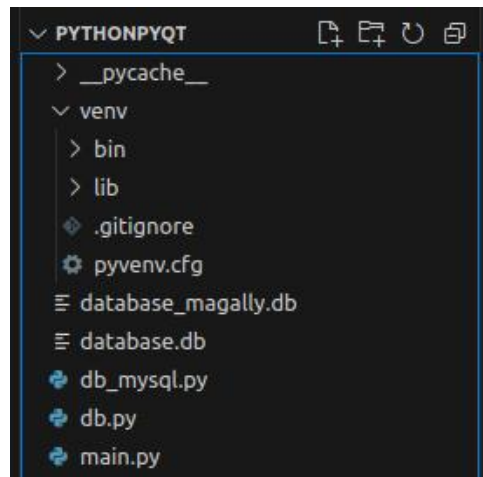
In this code, the class `TestCardUGTS(QFrame)` uses `QFrame` as a container for organizing the layout and `QTableWidget` to display and manage test data from the database. It also has other functions for sending the actual data if changed to backend and set the table as a changeable area for the user.

```
1 class TestCardUGTS(QFrame):
2
3 def __init__(self, test_id, scenario, test, cat, agr, orig, n_ugts,
4     objet_du_test, main_window):
5
6 self.main_window = main_window
7
8 self.test_id, self.scenario, self.test, self.cat, self.agr, self.orig,
9     self.n_ugts, self.objet_du_test = test_id, scenario, test, cat, agr,
10    orig, n_ugts, objet_du_test
11
12 self.setStyleSheet('background:white; border-radius:4px; color:black;')
13 layout = QHBoxLayout(self)
14 self.table = QTableWidget()
15
16 self.table.setStyleSheet('QTableView::item {border-right: 1px solid #
17     d6d9dc;}')
18
19 self.add_table_row(scenario, test, cat, agr, orig, n_ugts, objet_du_test
20 )
21
22 layout.addWidget(self.table)
23 self.setLayout(layout)
24
25 def add_table_row(self, scenario, test, cat, agr, orig, n_ugts,
26     objet_du_test):
27
28
29 def on_text_changed(self, row, column):
30 ...
31
32 def update_data(self, row, column, new_value):
33 ...
34
35 self.send_to_backend()
36
37 def send_to_backend(self):
38 ...
39
40 )
```

### 4.2.2.3 Repertory Organization

In the figure 19 show the main files used in the development of the Local Application, among them `__pycache__`: contains compiled `bytecode` of Python files, which helps in speeding up program execution. `database_magally.db`: A database file, in SQLite, used for storing application data. `database.db`: Another database file, in MySQL database. `main.py`: The main entry point of the application.

Figure 19 – VSCode Repertory View.



Source: Made by the Author.

## 4.3 DATABASE IMPLEMENTATION

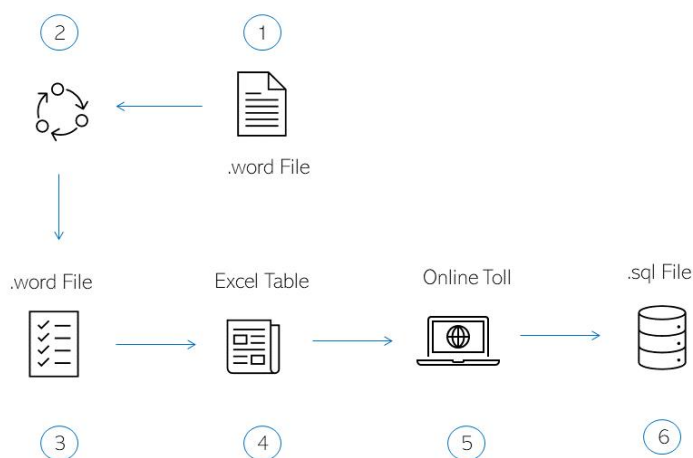
For creating the database for this project first was necessary a `.sql` file, an `.sql` file is a plain-text file that contains Structured Query Language (SQL) commands. These commands are used to perform operations on a database, such as creating tables, inserting data, querying information, updating records, or managing database schemas. They are highly portable and can be executed in various database systems including the one used for this project. The one used was translating the database structure, queries, and relationships from the Word document into SQL syntax. It includes `CREATE`, `INSERT`, `UPDATE`, and `DELETE` statements.

For the implementation of the `.sql` file with all the SQL requests already presents in the Microsoft `.word` file some steps were necessary as shown in figure 21.

The database was initially in the format of a table within a Word document, with each row representing a test, from it the following steps are taken::

1. Filter the original Word document so that it only contained the table, ensuring each row corresponded to a single test.

Figure 20 – Workflow creation sql file.



Source: Made by the Author.




2. Review the document to ensure the quality and accuracy of the table and its contents.
3. transfer all the tests to an Excel sheet.
4. After verifying the consistency of each test in the Excel sheet, was used an online tool to convert the comma-separated values (CSV) file into SQL queries, where each row generated an SQL insert statement to add the test into the database.

#### 4.3.1 Database Overview

The two tables in Figure 21 were used in this project:

- Scenario: The name of the associated scenario.
- Test: The name of the test.
- Orig: The origin of the scenario.
- Agr: The aggressive test indicator.
- Cat: The category of the scenario.
- N° UGTS: The number of the UGTS concerned by the test.
- Objet du test: A brief description of the test.

Figure 21 – UGE et UGTS table schema.

UGE		UGTS	
id 	integer	id 	integer
scenario	varchar	scenario	varchar
test	varchar	test 	text
agr	binary	agr	binary
cat	varchar	cat	varchar
orig	varchar	orig	varchar
N_UGTS	integer	N_UGTS	integer
objet_du_test	varchar	objet_du_test	varchar
created_at	timestamp	created_at	timestamp

Source: Made by the Author.

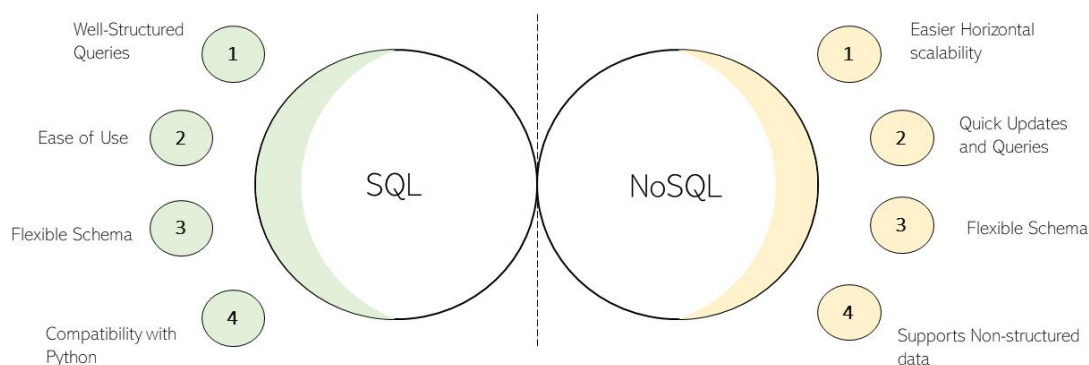
### 4.3.2 Relational Database Overview

SQL (Structured Query Language) and NoSQL (Not Only SQL) are two paradigms for managing and querying data in database systems. Their primary distinction lies in the structure and approach they take toward data organization and retrieval. SQL databases are relational database management systems (RDBMS) that use structured schemas to organize data into tables with rows and columns. SQL databases ensure Atomicity, Consistency, Isolation, and Durability, making them reliable for transactions. NoSQL databases are non-relational systems designed to handle unstructured or semi-structured data, offering flexible schemas. The terms "relational" and "non-relational" correspond to SQL and NoSQL databases, respectively.

While SQL databases emphasize structured relationships and query reliability, NoSQL systems focus on flexibility and scalability. They are complementary rather than exclusive, with many modern applications using a hybrid approach depending on the specific use case (ELMASRI; NAVATHE, 2016).

For this project was made the decision to use a relational database in detriment to a non-relational. The figure 22 illustrates the main differences between SQL and NoSQL databases, explaining why SQL was chosen as the most suitable option for this project over a NoSQL database. The primary reason is that SQL provides a simpler and more straightforward way to store the existing data.

Figure 22 – Differences SQL and NoSQL.



Source: Made by the Author.

Some main reasons that played an important role in the choice of SQL database:

- **Well-structured queries:** SQL databases use structured query language, making it ideal for complex data processing tasks.
- **Ease of use:** SQL is easy to learn and use for future implementation of the project.
- **Flexible schema:** SQL databases have a highly flexible schema that can manage various data types.
- **Compatible with popular programming languages:** SQL is compatible with Python, which is convenient in this project.

#### 4.3.2.1 MySQL database

MySQL is an open-source relational database management system (RDBMS) that uses Structured Query Language (SQL) for database operations. Developed in the mid-1990s, it has become one of the most widely used databases due to its reliability, speed, and ease of use as demonstrated and implemented in (LETKOWSKI, 2015).

The database management system selected for this project, MySQL, is a free and open-source software solution known for its adherence to a client-server architecture. The client-server architecture is a distributed system framework that divides tasks and workloads between service providers, known as servers, and service requester, called clients. This design is a cornerstone of modern computing, enabling scalable and efficient communication between users and centralized resources. This choice enables efficient communication and robust database operations (TANENBAUM; STEEN, 2011).



To connect to the MySQL database, the `mysql.connector` module was used, an official MySQL driver developed and maintained by Oracle, which allows Python applications to connect to and manage MySQL databases. This connector supports all MySQL features and can execute SQL queries, retrieve results, and manage transactions between Python and the database. The following code demonstrates how it was established the connection with the local MySQL server:

```
1
2 import mysql.connector
3
4 def connect_database():
5
6 mydb = mysql.connector.connect(
7
8 host = "localhost",
9 user = "root",
10 password = "password",
11 database = "test_1_magally"
12 )
13
14 mycursor = mydb.cursor()
15 return mycursor, mydb
```

### 1. Importing the `mysql.connector` module:

- The line `import mysql.connector` imports the MySQL Connector Python module, which allows Python to communicate with a MySQL database.

### 2. Defining the function `connect_database()`:

- `connect_database()` is a function defined to establish a connection to a MySQL database and return a cursor object and the database connection object.

### 3. Establishing a database connection:

- Inside the function, `mysql.connector.connect()` is used to connect to the MySQL database with the following parameters:
  - `host="localhost"` specifies that the MySQL server is located on the local machine.
  - `user="root"` specifies the username used to authenticate.
  - `password="password"` specifies the password for the user.
  - `database="test_1_magally"` specifies the name of the database to connect to.

- The resulting database connection object is stored in `mydb`.

#### 4. Creating a cursor object:

- `mydb.cursor()` creates a cursor object (`mycursor`) that allows Python code to execute SQL queries on the connected database.

#### 5. Returning the cursor and database connection objects:

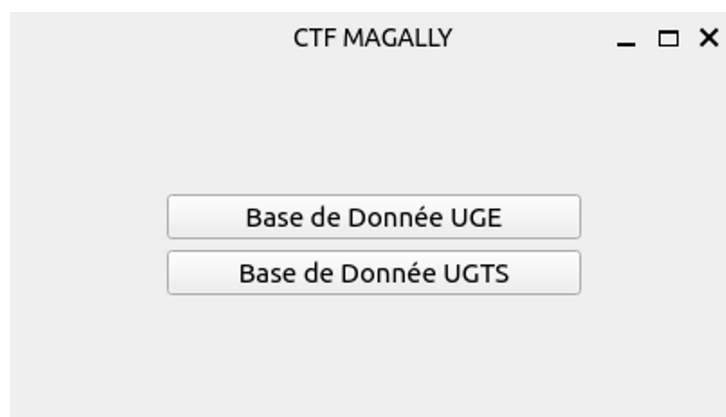
- Finally, the function returns `mycursor` and `mydb` as a tuple (`return mycursor, mydb`). This allows the calling code to use `mycursor` to execute SQL commands on the database and `mydb` to manage the database connection.

## 4.4 GUI DESIGN

For this section, some main features of the design will be shown for better comprehension of the application developed during this project.

The figure 23 demonstrates how the user can choose between both interfaces:

Figure 23 – UI to choose between databases.



Source: Made by the Author.

The figure 24 shows the search bar created and a search button:

Figure 24 – UI search bar.



Source: Made by the Author.

For adding new tests, Figure 25 shows the additional Window created for this purpose.

For deleting tests, the window demonstrated in Figure 26 was created:

Figure 25 – UI Add New Test.

Scenario: Test: Orig:

Cat: Agr: N° UGTS:

Objet du Test:

Objet Du Test

Add Test

Source: Made by the Author.

Figure 26 – UI Delete Test.

Delete Test

Nom du test à supprimer:

Nom Test

Supprimer

Source: Made by the Author.

And finally, the main window used for viewing all the tests related to the choose database can be seen in Figure 27:

Figure 27 – UI main window HMI Local App.



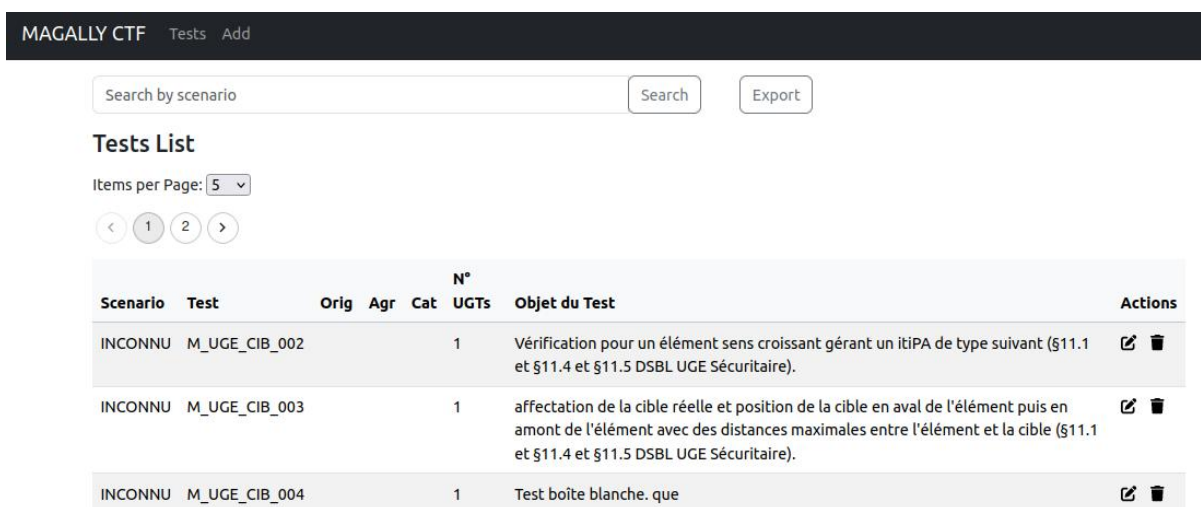
The screenshot shows a window titled "BD UGTS" with a search bar and buttons for "Search", "Clear", "Add Test +", "Delete Test", and "+ Version". Below the search bar is a table with the following data:

Scenario	Test	Cat	Agr	Orig	N_UGTS	Objet_du_Test
SCENARIO	UFEFE	X	none	3	1111	Vérification pour un élément téléchargé en garage, en place 3, se déplaçant dans le sens croissant : FU et ALSens sur AUCI garage décroissant, demande de retournement sur affectation sens sol décroissant d'ou ALSens désarmée sur affectation sens sol décroissant d'ou ALSens désarmée 11
I_007 (I_UGE_ITI_024)	0	0AA	1	0AA444	A777	Vérification pour un élément téléchargé en garage, en place 3, se déplaçant dans le sens croissant : FU et ALSens sur AUCI garage décroissant, demande de retournement sur affectation sens sol décroissant d'ou ALSens désarmée, AlRecul armée car sens RPH croissant, puis désarmée après dépassement du seuil d'inversion de sens. (§10.4.1 et §10.6 et §10.7 et §10.8 et §10.9 et §10.10 et §10.11 DSBL UGE Sécuritaire).
I_007 (I_UGE_ITI_024)	0	0	0	0	0	Vérification pour un élément téléchargé en garage, en place 3, se déplaçant dans le sens croissant :
I3 (M_UGE_LOC_602)	E_UGE_LOC_013	ASA	x	0	12 à 11	Vérification de la prise en compte du point d'arrêt d'entrée de section lorsque le décalage entre les limites de sections électriques et d'automatismes est constitué de deux segments, suite à une coupure HT alors que l'élément exécute son retournement après avoir franchi la limite de section électrique et gère son contrôle d'immobilisation






Source: Made by the Author.

The Web has a main window as shown in Figure 28, it is also possible to see the export button in the side of the search bar:

Figure 28 – UI main window HMI Web Application.



The screenshot shows a web interface with a header "MAGALLY CTF Tests Add". Below the header is a search bar with "Search by scenario" and buttons for "Search" and "Export". Below the search bar is a "Tests List" section with "Items per Page: 5" and pagination controls. The table below shows the test list:

Scenario	Test	Orig	Agr	Cat	N° UGTS	Objet du Test	Actions
INCONNU	M_UGE_CIB_002				1	Vérification pour un élément sens croissant gérant un itiPA de type suivant (§11.1 et §11.4 et §11.5 DSBL UGE Sécuritaire).	 
INCONNU	M_UGE_CIB_003				1	affectation de la cible réelle et position de la cible en aval de l'élément puis en amont de l'élément avec des distances maximales entre l'élément et la cible (§11.1 et §11.4 et §11.5 DSBL UGE Sécuritaire).	 
INCONNU	M_UGE_CIB_004				1	Test boîte blanche. que	 

Source: Made by the Author.

## 5 ANALYSIS OF RESULTS

This chapter summarizes the outcomes of the project and discusses the final solution adopted by the team. It evaluates the benefits and drawbacks of each solution considered during the project.

### 5.1 CYBER THREATS ANALYSIS

#### **Web Application Solution:**

Web applications are inherently exposed to a wider range of cyber threats due to their online accessibility. Key vulnerabilities and attack more prone to happen in the case of deployment of this project include:

- **Injection Attacks:** Web applications are particularly vulnerable to SQL injection, command injection, and similar attacks where malicious code is injected into user input fields. Poor validation and sanitization of user inputs exacerbate this risk (ALGHAWAZI; ALGHAZZAWI; ALARIFI, 2022).
- **Cross-Site Scripting (XSS):** Attackers can exploit vulnerabilities to inject malicious scripts into web pages viewed by users, compromising user data or session tokens.
- **Denial of Service (DoS) and Distributed Denial of Service (DDoS):** Overloading a web application server with traffic can render the service unavailable to legitimate users (ALI; CHONG; MANICKAM, 2023).

To address these issues, further enhancements should be made to the developed solution, ideally with the expertise of a cybersecurity engineer. As outlined in (STUTTARD; PINTO, 2011), effective mitigation strategies include adopting secure coding practices such as input validation and parameterized queries, implementing strong authentication and authorization mechanisms like OAuth or multifactor authentication, and utilizing Web Application Firewalls (WAFs) to detect and block malicious traffic.

By ensuring the rigorous application of these strategies, the web application can achieve a high standard of security, protecting users and safeguarding sensitive client data.

### 5.2 ADVANTAGES AND DISADVANTAGES OF EACH SOLUTION

#### **Web Application Solution:**

*Advantages:*

- **User Interface:** Utilizing React, a modern JavaScript library, enables the creation of dynamic and responsive user interfaces. The vast ecosystem of libraries and tools available for React makes it easier to enhance user experience and interface performance. React's component-based architecture promotes reusability, ensuring cleaner and more maintainable codebases. Its robust ecosystem of tools and libraries, such as Redux for state management and React Router for navigation, simplifies the implementation of advanced functionality.
- **Scalability:** Web applications can be easily scaled to accommodate more users or increased data loads. This is particularly beneficial if the project grows in requirements, and expects a large users base.
- **Ease of Deployment:** The Web application can be seamlessly deployed using containerization technologies like Docker presented in chapter 5. This simplifies the deployment process, allowing for consistent environments across different stages of development and production.

*Disadvantages:*

- **Security Concerns:** The web applications requires robust security measures to protect it against threats such as data breaches, unauthorized access, and cyber attacks. Proper authentication, encryption, and other security protocols must be implemented and maintained.
- **Higher Deployment Costs:** Deploying the web application involved additional expenses, such as hosting fees, domain registration, and maintaining all the server infrastructure.
- **Maintenance Complexity:** Managing the web application after deployment can be complex, necessitating a dedicated team to handle updates, bug fixes, and potential issues that would probably arise in the live environment.

### Local Application Solution:

#### *Advantages:*

- **No Web Deployment Costs:** The Local application does not incur in costs associated with web deployment, such as server hosting.
- **Maintenance:** Local applications, often simpler in nature, can be easier for developers to understand and maintain. This can lead to faster development cycles and easier troubleshooting.
- **Data Security:** Sensitive client information is not transmitted over the internet, reducing the risk of data breaches. All data remains within the user's local system, providing an added layer of security.
- **No Authentication Required:** The Local application do not require user authentication or management of user accounts, simplifying the development process and reducing potential security vulnerabilities.

#### *Disadvantages:*

- **Limited Libraries for User Experience:** PyQt, the framework used for the local application, has fewer libraries and tools available for enhancing user experience compared to web frameworks like React. This resulted in a less polished interface.
- **Complexity in Implementing Features:** Implementing certain features, such as exporting data to an Excel file, was much more complex in the local application. Web applications often have more straightforward solutions and third-party libraries available for such tasks. Local applications may require additional steps to set up dependencies or manage file system interactions manually, especially if the deployment environment varies across systems. In web applications, tasks like exporting data often have established patterns and community-supported libraries or services, which can make implementation simpler.

## 5.3 OVERVIEW OF PREFERRED SOLUTION

The team opted for the Local Application solution primarily due to cybersecurity concerns and cost considerations. Handling sensitive client data was a critical factor in this decision. Even with robust authentication mechanisms, exposing such data on a web platform posed significant security risks. By deploying the application locally, we ensured that all sensitive information remained within the client's environment, thereby mitigating potential cyber threats.

Additionally, the Local Application solution proved to be more cost-effective. Avoiding web deployment eliminated the need for server hosting, domain registration,

and other associated expenses. This approach not only reduced costs but also simplified the deployment process, making it quicker and easier to implement the solution within the client's existing infrastructure.

Overall, the combination of enhanced data security and lower deployment costs made the Local Application solution the most suitable choice for the project.



## 6 CONCLUSION

Most of the functional requirements for this project, as outlined in Chapter 3.2.1.1, were successfully implemented in both the Web Application and the Local Application solutions.

The team expressed high satisfaction with the demonstration of both solutions, particularly as they adhere to prevailing market trends in database management. MySQL, being a widely adopted open-source database management system, is recognized for its scalability, reliability, and extensive community support, making it a preferred choice across industries (MOHAN et al., 2013). The user interfaces were also deemed well-structured and appropriate for the project's scope, offering a user-friendly experience in both implementations as exposed in chapter 4.4.

The decision to not deploying the Web Application on a hosting server was driven by the preference to keep the application local. This choice was primarily influenced by cybersecurity concerns, ensuring sensitive client data remains protected.

The database will remain internal to the company, stored on one of its on-premise cloud solutions. This approach aligns with the need to safeguard data from potential cyber threats associated with web hosting.

Overall, the Local Application solution was meticulously documented and stored for future development and potential deployment as the team's needs evolve. During the testing phase, the project demonstrated its usefulness to the MAGALLY team, significantly reducing errors and the time required to complete the tasks performed by the user presented in chapter 3.2.1.1.

The successful implementation of this project underscores the importance of prioritizing cybersecurity and cost-efficiency, leading to a robust and reliable Local Application solution.

## REFERENCES

ALGHAWAZI, Maha; ALGHAZZAWI, Daniyal; ALARIFI, Suaad. Detection of SQL Injection Attack Using Machine Learning Techniques: A Systematic Literature Review. **Journal of Cybersecurity and Privacy**, v. 2, n. 4, p. 764–777, 2022. ISSN 2624-800X. DOI: 10.3390/jcp2040039. Available from: <https://www.mdpi.com/2624-800X/2/4/39>.

ALI, Tariq Emad; CHONG, Yung-Wey; MANICKAM, Selvakumar. Machine Learning Techniques to Detect a DDoS Attack in SDN: A Systematic Review. **Applied Sciences**, MDPI, v. 13, n. 5, p. 3183, 2023. DOI: 10.3390/app13053183.

API - What is an API? [S.l.: s.n.]. Available from: <https://www.mulesoft.com/resources/api/what-is-an-api/>. 2024.

AXIOS - Promise based HTTP client for the browser and node.js. [S.l.: s.n.]. Available from: <https://axios-http.com/docs/intro/>. 2024.

CAPGEMINI ENGINEERING - Get the future you want : le future que vous voulez. [S.l.: s.n.]. Available from: <https://www.capgemini.com/fr-fr/notre-groupe/nous-connaître/>. 2024.

CHUNG, Lawrence; NIXON, Brian A.; YU, Eric; MYLOPOULOS, John. **Non-Functional Requirements in Software Engineering**. [S.l.]: Springer, 2012.

COCKBURN, Alistair. **Writing Effective Use Cases**. [S.l.]: Addison-Wesley Professional, 2000.

CONTAINERS - Containers are an abstraction at the app layer that packages code and dependencies together. [S.l.: s.n.]. Available from: <https://www.docker.com/resources/what-container/>. 2024.

DOCKER - Docker helps developers bring their ideas to life by conquering the complexity of app development. [S.l.: s.n.]. Available from: <https://www.docker.com/>. 2024.

DOCKER COMPOSE - Docker Compose is a tool for defining and running multi-container applications. It is the key to unlocking a streamlined and efficient

development and deployment experience. [S.l.: s.n.]. Available from:  
<https://docs.docker.com/compose/>. 2024.

ELMASRI, Ramez; NAVATHE, Shamkant. **Fundamentals of Database Systems**. 7th. [S.l.]: Pearson, 2016. ISBN 9780133970777.

EXPRESS - Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications. [S.l.: s.n.]. Available from: <https://expressjs.com/>. 2024.

FOWLER, Martin. **Patterns of Enterprise Application Architecture**. Boston, MA: Addison-Wesley, 2002. ISBN 978-0321127426.

FOWLER, Martin. **UML Distilled: A Brief Guide to the Standard Object Modeling Language**. [S.l.]: Addison-Wesley Professional, 2004.

GITLAB - From planning to production, GitLab brings teams together to shorten cycle times, reduce costs, strengthen security, and increase developer productivity. [S.l.: s.n.]. Available from: <https://gitlab.com/>. 2024.

HTTP - HTTP is an application layer protocol designed to transfer information between networked devices and runs on top of other layers of the network protocol stack. [S.l.: s.n.]. Available from:  
<https://www.cloudflare.com/learning/ddos/glossary/hypertext-transfer-protocol-http/>. 2024.

JAVASCRIPT - JavaScript (JS) is a lightweight interpreted (or just-in-time compiled) programming language with first-class functions. [S.l.: s.n.]. Available from:  
<https://www.javascript.com/>. 2024.

KEOLYS - Leaders Mondiaux de la Mobilité Partagée. [S.l.: s.n.]. Available from:  
<https://www.keolis.com/>. 2024.

LETKOWSKI, Jerzy. Doing database design with MySQL. **Journal of Technology Research**, Volume 6, Jan. 2015.

LINUX - Just like Windows, iOS, and Mac OS, Linux is an operating system. In fact, one of the most popular platforms on the planet, Android, is powered by the Linux

operating system. [S.l.: s.n.]. Available from:  
<https://www.linux.com/what-is-linux/>. 2024.

MAGALLY - Métro Automatique à Grand Gabarit de l'Agglomération LYonnaise. [S.l.: s.n.]. Available from: <https://www.techno-science.net/definition/15570.html/>. 2024.

MERKEL, Sean. **Docker: Up Running: Shipping Reliable Containers in Production**. [S.l.]: O'Reilly Media, 2014. ISBN: 978-1491917572.

MES - Manufacturing Execution System. [S.l.: s.n.]. Available from:  
<https://www.sap.com/france/products/scm/execution-mes/what-is-mes.html/>. 2024.

MICROSOFT EXCEL - Analysez, comprenez et visualisez vos données en toute simplicité. [S.l.: s.n.]. Available from:  
<https://www.microsoft.com/fr-fr/microsoft-365/excel/>. 2024.

MOHAN, Chandan et al. Open source databases: An opportunity for effective database management in the cloud era. **International Journal of Computer Applications**, Foundation of Computer Science, v. 80, n. 14, 2013.

MYSQL - The world's most popular open source database. [S.l.: s.n.]. Available from:  
<https://www.mysql.com/>. 2024.

NEGUS, Christopher. **Linux Bible: Everything You Need to Know About the Linux Operating System**. 9th. [S.l.]: Wiley, 2014. ISBN: 978-1118999875.

NODE - Node.js® is a free, open-source, cross-platform JavaScript runtime environment that lets developers create servers, web apps, command line tools and scripts. [S.l.: s.n.]. Available from: <https://nodejs.org/en/>. 2024.

PIPENV - Python Development Workflow for Humans. [S.l.: s.n.]. Available from:  
<https://pypi.org/project/pipenv/>. 2024.

PRESSMAN, Roger S. **Software Engineering: A Practitioner's Approach**. 8th. [S.l.]: McGraw-Hill Education, 2014. P. 123–125. Discusses the importance of functional requirements in project development. ISBN 978-0078022128.

PYQT - PyQt is one of the most popular Python bindings for the Qt cross-platform C++ framework. [S.l.: s.n.]. Available from:

<https://www.riverbankcomputing.com/static/Docs/PyQt5/>. 2024.

PYTHON - Python is a programming language that lets you work quickly and integrate systems more effectively7. [S.l.: s.n.]. Available from: <https://www.python.org/>. 2024.

QT DESIGN STUDIO - Revolutionize your development process by bridging the gap between designers and developers to turn your design visions into production-ready UIs. [S.l.: s.n.]. Available from: <https://www.qt.io/product/ui-design-tools/>. 2024.

REACT - A JavaScript library for building user interfaces. [S.l.: s.n.]. Available from: <https://reactjs.org/>. 2024.

REDUX - A JS library for predictable and maintainable global state management. [S.l.: s.n.]. Available from: <https://redux.js.org/>. 2024.

REDUX STORE - The Redux store brings together the state, actions, and reducers that make up your app. [S.l.: s.n.]. Available from: <https://redux.js.org/tutorials/fundamentals/part-4-store/>. 2024.

REST API - What is a REST API? [S.l.: s.n.]. Available from: <https://www.redhat.com/en/topics/api/what-is-a-rest-api/>. 2024.

SARISHMA, Abhishek. A Systematic Review of the Impact of Containerization on Software Development and Deployment Practice. **Journal of Applied Computer Science and Intelligent Technologies**, 2021. DOI: 10.17492.

SEQUELIZE - Featuring solid transaction support, relations, eager and lazy loading, read replication and more. [S.l.: s.n.]. Available from: <https://sequelize.org/>. 2024.

SQL - Structured Query Language. [S.l.: s.n.]. Available from: <https://sql.sh/>. 2024.

STUTTARD, Dafydd; PINTO, Marcus. **The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws**. 2nd. [S.l.]: Wiley, 2011. ISBN 978-1118026472.

SYSML - Systems Modeling Language. [S.l.: s.n.]. Available from:  
<https://sysml.org/>. 2024.

SYTRAL - Le Sytral Mobilités est un établissement public qui a en charge l'organisation et l'exploitation des transports en commun urbains de l'agglomération lyonnaise. [S.l.: s.n.]. Available from: <https://www.sytral.fr/>. 2024.

TANENBAUM, Andrew S.; STEEN, Maarten Van. **Distributed Systems: Principles and Paradigms**. [S.l.]: Pearson Education, 2011. ISBN 978-0132143010.

TCL - Transports en commun à Lyon. [S.l.: s.n.]. Available from:  
<https://www.tcl.fr/>. 2024.

TEAM, Tabnine. **Is Visual Studio Code really the best code editor?** [S.l.: s.n.], 2023. Available at <https://www.tabnine.com/blog/visual-studio-code/>. Available from:  
<https://www.tabnine.com/blog/visual-studio-code/>.

UBUNTU - Ubuntu is the modern, open source operating system on Linux for the enterprise server, desktop, cloud, and IoT. [S.l.: s.n.]. Available from:  
<https://ubuntu.com/>. 2024.

V Model - The V-model is a graphical representation of a systems development lifecycle. [S.l.: s.n.]. Available from:  
<https://www.geeksforgeeks.org/software-engineering-sdlc-v-model/>. 2024.

VISUAL Studio Code - Visual Studio Code is a code editor redefined and optimized for building and debugging modern web and cloud applications. [S.l.: s.n.]. Available from:  
<https://code.visualstudio.com/>. 2024.

WINDOWS 11 - Windows 11 is the latest major release of Microsoft's Windows NT operating system, released on October 5, 2021. [S.l.: s.n.]. Available from:  
<https://learn.microsoft.com/en-us/windows/whats-new/windows-11-overview/>. 2021.