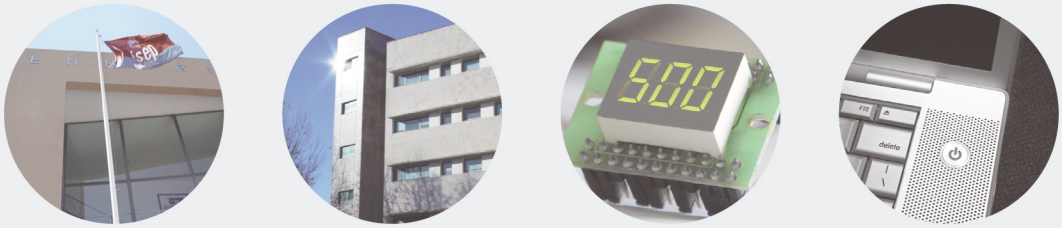
INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES



Ferramenta de Suporte para Apresentação e Gestão de Dados Recolhidos por Sistemas Autónomos

RAIMUNDO ANTÓNIO SÁ E SILVA
Outubro de 2020

POLITÉCNICO
DO PORTO

Support Tool for Visualization and Management of Autonomous System Data

Mestrado em Engenharia Eletrotécnica e de Computadores

Raimundo Silva

1150521@isep.ipp.pt

Instituto Superior de Engenharia do Porto

Acknowledgments

First and foremost, I would like to thank my family and friends for sharing this journey with me so far, I couldn't have done it without your love and support. You are the reason I keep moving forward.

I would also like to extend my gratitude to my advisor, Nuno Dias, for his incredible availability and patience with me. I will forever be grateful for the help provided over the past few months, and will always be inspired by your diligence.

To all the teachers I had during my masters degree that believed in me, I give you my sincerest of gratitude. I know I was never the perfect student, but the example that was set by you was a demonstration on why I should always strive to be the best I can be.

This page was intentionally left blank.

Resumo

À medida que as tecnologias relacionadas com os sistemas autónomos progridem, a quantidade de informação obtida dos mesmos também progride. Os sensores que constituem estes sistemas estão a tornar-se progressivamente mais eficientes e a cooperação entre grupos de sensores torna-se cada vez mais complexa.

Tudo isto justifica a necessidade da existência de um sistema de uso simples, o qual obtém dados de um número elevado de sensores ou grupo de sensores e os interpreta e visualiza de forma imediata.

Uma solução a esta necessidade deve procurar adquirir informação providenciada por sensores associados a sistemas autónomos, seja esta informação obtida através de ficheiros de *log* ou através de comunicação direta através de *sockets*, seguida da escrita para bases de dados da informação interpretada.

A ferramenta de manipulação de bases de dados, desenvolvida em *Python*, utiliza "MySQL" *queries* de forma a criar, alterar ou apagar tabelas dentro de uma base de dados, tabelas as quais serão preenchidas com os dados interpretados a partir de *logs* ou comunicação com sensores em tempo real, via *sockets*. Por outro lado, a ferramenta de visualização *online*, desenvolvida em HTML (Hypertext Markup Language), PHP (PHP: Hypertext Preprocessor) e *JavaScript*, utiliza PHP de forma a obter dados através de "MySQL" *queries* e utiliza gráficos e outras ferramentas visuais, disponibilizadas através de *JavaScript*, de forma a demonstrar os dados obtidos de forma simplificada.

Palavras-chave

Sistemas, Sensores, Visualização, Armazenamento, Ferramenta, Dados, *Online*, *Python*, HTML, *JavaScript*, PHP.

This page was intentionally left blank.

Abstract

As the technology of autonomous systems evolves with the passage of time, so does the amount of information we can obtain from them. The sensors that are part of these systems are becoming progressively more efficient and cooperation between groups of sensors is becoming ever more complex.

All of this creates a need for a simple-to-use system, that obtains sensor data from a plethora of different sensors or groups of sensors and parses and visualizes them in an immediate fashion.

A solution to this need should aim to acquire information provided by sensors associated with autonomous systems, be that information obtained from previously existent log files or through direct communication utilizing sockets, followed by the writing to databases of the interpreted information. Once that information is acquired, interpreted and stored, it should then be easy to visualize and access on any platform.

The database manipulation tool, developed in *Python*, uses "MySQL" queries in order to create, alter or delete tables inside a database, tables which will be filled with the interpreted data obtained from logs or real-time sensor communication. On the other hand, the online visualization tool, developed in HTML (Hypertext Markup Language), PHP (PHP: Hypertext Preprocessor) and *JavaScript*, uses PHP coding to obtain data through "MySQL" queries and uses graphs and other visual tools, available through *JavaScript*, to convey the acquired data in a simplified fashion.

Keywords

Systems, Sensors, Visualization, Storage, Tool, Data, Online, *Python*, HTML, *JavaScript*, PHP.

This page was intentionally left blank.

Contents

This page was intentionally left blank.

List of Figures

List of Tables

This page was intentionally left blank.

List of Acronyms

API *Application Programming Interface*

AI *Artificial Intelligence*

BI *Business Intelligence*

BSD *Berkeley Software Distribution*

CSS *Cascading Style Sheets*

CSSE *Center for Systems Science and Engineering*

CSV *Comma-Separated Values*

CTD *Conductivity, Temperature and Depth*

DNA *Deoxyribonucleic Acid*

EEA *European Environment Association*

EPL *Eclipse Public License*

GMT *Greenwich Mean Time*

GNSS *Global Navigation Satellite System*

GPL *General Public License*

GPS *Global Positioning System*

HTML *HyperText Markup Language*

HTTP *Hypertext Transfer Protocol*

IDPL *Initial Developer's Public License*

INESC TEC *Instituto de Engenharia de Sistemas e Computadores*

IoT *Internet of Things*

IP *Internet Protocol*

IPL *InterBase Public License*

ISEP *Instituto Superior de Engenharia do Porto*

JHU *John Hopkins University*

LAMP *Linux, Apache, MySQL, PHP*

MPL *Mozilla Public License*

MVCC *Multiversion Concurrency Control*

PHP *PHP: Hypertext Preprocessor*

PSU *Practical Salinity Units*

RDBMS *Relational Database Management Systems*

ROS *Robot Operating System*

RNA *Ribonucleic Acid*

SAIL *Space-Atmosphere-Ocean Interactions in the marine boundary Layer*

SONAR *Sound Navigation and Ranging*

URL *Uniform Resource Locator*

USB *Universal Serial Bus*

Chapter 1

Introduction

1.1 Contextualization

Nowadays and given the accelerated rate of evolution for new technologies, the incorporation of sensors in autonomous systems is becoming ever more complex. As such, the information we obtain from these autonomous systems is also becoming larger in scale, creating a need to develop systems which observe and simplify the obtained information before it is presented to an user. That being said, adequately storing the data acquired from these autonomous systems is also of high importance, given that large amounts of data is hard to store in an organized and simple fashion. Taking that into account, the development of a tool or set of tools capable of both visualizing and storing data is a necessity in this day and age.

It is important to mention that, however, some autonomous systems have their own associated software to visualize and possibly store all of the obtained data. The problem is that these software solutions are restricted to one use case, that being of their associated autonomous system, and are not generally applicable. Having a software solution that can store and visualize data from any autonomous system that outputs data in a general sense, even those that have a proprietary software packaged with them,

would be a true facilitator when it comes to the day-to-day activities of any user that manages multiple autonomous systems.

In that regard and in the vein of the volume of data that can be acquired from autonomous systems, certain filtering and customization options for the stored and observed data would be an interesting proposition. For example, if a certain user wants to observe whether or not there is a problem with a machine at a glance, having a solution at their disposal that contains strong filtering and customization features would facilitate that process. A solution that is capable of associating certain sensors with one another and presents their data in tandem whilst ignoring other sensors that are irrelevant to the task at hand, depending on any users' wishes and regardless of system used, would make it so that the information clutter is minimized and vital decision making can be made immediately. This would make it so that having to sort through a large amount of useless data for an issue at hand would no longer be a necessity.

1.2 Objectives

The main objective behind this thesis is to create a solution for the storage and visualization of log and real-time data of any sensor or group of sensors that output data.

When it comes to storage, Relational Database Management Systems (RDBMS) are a great way for safe and easy-to-access data storage and should be utilized to their fullest potential.

Having this in consideration, the body of work described in this thesis will focus around the creation of a data management solution, which will include both data storage and visualization components, that is easy to use and access by anyone.

The overall solution should also be applicable to any sensor/group of sensors regardless of whether or not the development of the project they are

being used for has been concluded. As such, not only can the solution aid in managing the data of a finished product, but also help with the testing processes associated with the development of that product.

1.3 Problem Description

Nowadays, autonomous systems have advanced to a point in which almost any characteristic of a certain environment, be it physical or chemical, can be observed and detailed through their usage. This is done through complex sensor interfaces that acquire data that can then be processed in the context of its importance, processing which can be conducted both inside or outside the autonomous system itself.

When exporting data for processing outside of an autonomous system, it can be done through direct communication between the autonomous system and the processes running inside a network, in the case that the autonomous system has a direct communication system embedded into itself. If the autonomous system responsible for data output does not include any sort of direct network communication features, the data can be acquired manually by taking it directly from an autonomous system into an auxiliary computer capable of running the processes, once that autonomous system is accessible.

Finally, if the autonomous system has the capacity of running the necessary processes itself, which normally entails the existence of an operating system, it should be capable of parsing its data internally.

A general view of different possible problem scenarios an autonomous system, attempting to communicate with a network, can find itself in is represented in Figure 1.1.

By analyzing the provided overall problem scenarios, we can see some of the different situations which autonomous systems could find themselves in when sending data.

In scenario 1, the autonomous system is inside a mine and, although

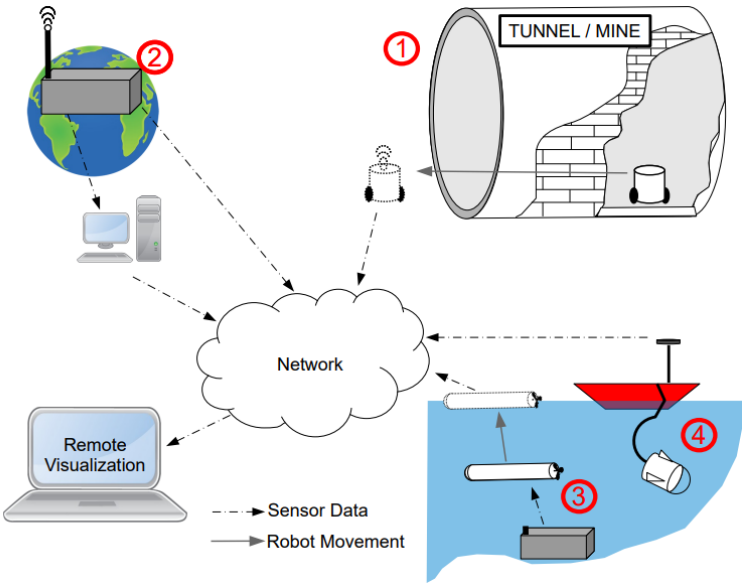


Figure 1.1: Possible Problem Scenarios.

it does have the capability of sending data directly to a network, it needs first to move out of the mine, either autonomously or manually with human assistance, so that it can establish communication with the network. This is due to the fact that wireless reception is not possible in places such as mines, and a clearing needs to be found in order to establish wireless communication of any sort. Another way of circumventing this issue would be to setup a wired communication between an autonomous system inside the mine and a computer outside the mine that can then be used to send data through a network instead.

When it comes to scenario 2, the autonomous system has the capability of communicating directly to the network to any place in the planet, either by utilizing an auxiliary computer or by communicating directly with the network that writes/reads databases.

Lastly, in the case of scenarios 3 and 4, the autonomous systems are underwater, and as such do not have any way of communicating wirelessly to the outside world. To solve this problem, one of two options can be

used, either have another system (be it autonomous or manned) go to the localization of the autonomous system and access its data, so the data can then be sent by the system to the network once it reaches the surface (as seen in scenario 3) or connect a cable between the autonomous system and an auxiliary computer capable of connecting to the desired network (as seen in scenario 4).

Accordingly and addressing the main issue that will be tackled in this thesis, all the data that is taken from the sensor interfaces needs to be made comprehensible by any user that requires that data to draw conclusions regarding the environmental situation that surrounds the autonomous system.

As also mentioned, certain autonomous systems can process the data and introduce it to their users, as well as external tools that do the exact same thing. The problem with those tools and systems is that they are commonly very case specific, meaning that one tool/system cannot be utilized for a plethora of different sensor interfaces' data. Solving this problem, by creating a generally applicable data processing tool, would greatly facilitate all sorts of autonomous system data acquisition and interpretation to any user.

1.4 Requirements

In order to be a successful implementation in the context of the problem, the conceptual solution should include the following characteristics:

- Interpretation and storing of data from a multitude of different sources;
- Data visualization with strong customization and filtration options;
- Able to read databases and write into them;
- Widely accessible by those with permission;
- Capable of obtaining data both current and old;

- Configurable missing data (if necessary).

1.5 Use Cases

The main concept behind the desired solution consists in a generally applicable data visualization tool, which means that any sort of obtainable data from sensors or group of sensors should be usable by the solution. Through this, the solution aims to facilitate decision making by its users, providing properly parsed data on a platform that's easy to access for any member of a team/group responsible for its related project. This also entails customization and filtration of important data, given the users' preferences at any time.

The use cases of the described solution could be, for example:

- Environmental characteristic observation and display;
- Geographical data parsing and positioning;
- Bio-diversity supervision and control.

There are 3 autonomous systems/projects that encapsulate perfectly the use cases for the presented concept. These are the "MarinEYE" and "Feed-First" autonomous systems and the Space-Atmosphere-Ocean Interactions in the marine boundary Layer ("SAIL") project [14].

The "MarinEYE" is an autonomous system that generates physical, chemical and biological data sets synchronized in time and space, utilized in the context of the ocean [15]. All the sensors included in this autonomous system can be seen in Figure 1.2.

As previously mentioned and has seen in Figure 1.2, not only are there a plethora of different sensors to take into consideration in this autonomous system, but most of these sensors are associated in larger systems in order to obtain data befitting of their purpose. For example, in the case of the

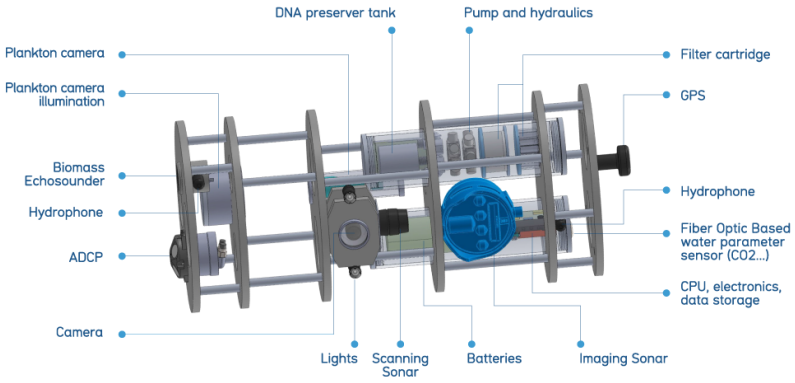


Figure 1.2: "MarinEYE" Concept [1].

hydroacoustic system, which is used to observe the biodiversity of undersea predators in the surrounding area of the "MarinEYE", a Sound Navigation and Ranging (SONAR) and hydrophone are utilized simultaneously in order to make assessments regarding identification and quantification of the predatory animals that come in contact with the "MarinEYE" [15].

In order to develop a successful solution applicable to the "MarinEYE", it is necessary that all sensors and their dependencies are taken into consideration. If systematic sensor association isn't respected and all the obtained data is simply demonstrated individually for each sensor, it could lead to erroneous conclusions by observers, given that some of the obtained information simply does not make sense when observed individually.

The "SAIL" project consists in an undergoing mission, whose main goal is to measure the planet's atmospheric electric field and collect information that has not been updated for more than 100 years, having been acquired only by the ship *Carnegie*, between 1909 and 1920 [16].

To do this, the project is being conducted in the school-vessel *Sagres* during its world tour, and utilizes an interface between a multitude of different sensors in tandem to achieve the previously mentioned goal. Some of these sensors were associated with the ship itself, however most of the sen-

sors are kept in an autonomous system called "TowFish". The school-vessel includes multiple sensors working above water, ranging from electric field sensors to a Global Navigation Satellite System (GNSS), and, in addition to those sensors, the developed "TowFish" is used to obtain data underwater, observing data such as oxygen levels, salinity, temperature, etc. [12][13][16].

If a data storage and visualization tool were to be developed as a way to assist in the project's goals, it would need to be able to associate geographical data with the data output from all the different sensors acquired for the duration of the project.

Finally, the "FeedFirst" autonomous system is a solution developed with the intent of improving the way fish larvae reared in aquaculture are fed. This solution includes high-quality inert microdiets and a rearing tank that provides innovative hydraulic features [17].

In order to achieve the main goal of the solution, a very careful observation of characteristics such as temperature, salinity and oxygen levels need to be tracked, so that a safe growing environment for the larvae can be assured. Thus, the development of a data storage and visualization tool that is capable of collecting all these important characteristics and showing them to the end user in a quick, easy to interpret fashion is fundamental to the success of the "FeedFirst" autonomous systems' implementation.

1.6 Thesis Structure

The first chapter of this thesis is dedicated to the introduction of the problem statement, to be solved later on. The second chapter consists in the state of the art, which includes a theoretical study of all the concepts and market options that should be taken into consideration when developing a solution to the given problem. The third chapter is used for the analysis of requirements that a possible data storage and visualization tool needs to abide by in order to be successful in solving the given problem. In the fourth

chapter, the development of the overall solution itself will be detailed. The fifth chapter includes a demonstration of results of the finalized solution, through examples and a full application scenario. Finally, in the sixth chapter, a short conclusion will be made to the overall thesis, along with possible future improvements to the developed solution.

This page was intentionally left blank.

Chapter 2

State of the Art

In this chapter, a study of data storage and visualization options currently available in the market will be conducted.

The study of data storage technologies will be focused on data management systems, which will include data management application development tools and currently available market solutions.

When it comes to data visualization options, the study will focus around browser-based data visualization solutions, in order to determine the best way of creating accessible and user-friendly data visualization interfaces.

In order to provide fundamental understanding on how new data storage and web data visualization solutions could be developed, some of the coding language options available for the development of those solutions will be detailed.

Lastly in this chapter, some of the possible structures of sensor data output will be described, in order to better understand the parameters to take into consideration when developing sensor data storage and visualization tools.

2.1 Data Management Systems

This thesis is contextually applied to sensor data and, generally, this type of data can be stored either locally or in databases. Since part of the intention behind this thesis is to create a solution that can store sensor data in a way that can be easily accessible by anyone, databases should be the focus of solution development.

To develop these types of solutions, it is necessary to have some way of generating databases to store data into. To do this, RDBMS are necessary and, in this section, a comprehensive study of features will be made about them. By the end of the study, the best RDBMS will be chosen for solution development in the given context.

Lastly in this section, fully developed market solutions for data storage will be mentioned, in order to provide fundamental knowledge on what solutions exist and to better understand what they should be capable of.

2.1.1 Relational Database Management Systems

In this section, a comparison between RDBMS will be made, in order to determine which is the best option for database management given the contextualized problem and objectives. Also, by understanding the currently available options of database management to be used in the development of a solution, it facilitates the study of the market in regards to solutions that include their own database management tools, some of which will be described in the next section. There are a plethora of different RDBMS, but only the ones that are open-source will be mentioned, in order to limit systems whose inner workings are relatively unknown to the public.

The comparison between open-source RDBMS is represented, based on their basic characteristics, in Table 2.1.

RDBMS	Operating System Support	License
CUBRID	Windows/Linux	GPL V2
INGRES	Windows/MacOS/Linux	GPL V2
FireBird	Windows/MacOS/Linux	IPL and IDPL[18]
HSQLDB	Windows/MacOS/Linux	BSD
H2	Windows/MacOS/Linux	EPL
LucidB	Windows/MacOS/Linux	GPL V2
MariaDB	Windows/MacOS/Linux	GPL V2 [19]
MySQL	Windows/MacOS/Linux	GPL V2 or Proprietary[20][21]
MonetDB	Windows/MacOS/Linux	MPL
sqLite	Windows/MacOS/Linux	Public Domain[22]

Table 2.1: Relational Database Management Systems Comparison (Basic Features).

Licenses and Accessibility

Observing Table 2.1, we can see that there are multiple different types of licenses when it comes to the software of RDBMS.

The fundamental open-source license is General Public License (GPL, both V1 and V2), which provides legal permission to copy, distribute and/or modify the software protected by it [23].

In the same vein, the Berkeley Software Distribution (BSD) [24], Eclipse Public License (EPL), Initial Developer's Public License (IDPL), InterBase Public License (IPL) [18] and Mozilla Public License (MPL) [25] provide users with the freedom to copy, distribute and/or modify the software they encapsulate, being variation of the GPL license.

These licenses, however, all have their own (despite minimal) restrictions to the original GPL formula, and as such, for the sake of simplicity and ease of use, it's best to focus on RDBMS that are licensed through GPL in order to develop solutions, if a choice can be made.

On the other hand, there are some RDBMS, listed in Table 2.1, that have special licenses (or lack there of) that should be mentioned.

The "SQLite" relational database management system does not have a

license, so users are free to manipulate, share and copy the software in any way shape or form of their choosing. Restrictions only apply when it comes to contribution to the official version of the product itself. This means that it has just as much merit to be used in the development of solutions as RDBMS covered by GPL [22].

The "MySQL" relational database management system is one of the most commonly used open-source RDBMS, however it was acquired by "Oracle" a few years ago and, as such, some of their policies changed accordingly. Currently, "MySQL" has multiple versions, and only one of those is free and under GPL, all the others are proprietary and need to be paid for. As to be expected, the free version of "MySQL" has the least features, but can be modified, shared and copied at will [20][21].

Lastly, the FireBird relational database management system utilizes 2 licenses simultaneously: IPL and IDPL. The first one covers the parts of the source code that were inherited from the "InterBase" relational database management system and the second applies to the additions and improvements made by the "FireBird" Project. Despite this, the overall license still functions similarly to GPL, being a free open-source product, however it does have its own set of restriction like many other GPL-derivative licenses [18].

Database Locking

In order to limit the amount of information sent from a database to the end user upon request to fit their needs, data is stored in what is called "Locks". That being said, the amount of data varies between RDBMS and that's where the "granularity" of a lock comes into play. The "granularity" of a lock dictates the amount of information that is stored within it, going from anywhere between "coarse granularity", which means that each lock contains a large amount of data, and "fine granularity", which signifies that

each lock contains a small amount of data. Most of the time and as an example, if a lock has fine granularity, each one will correspond to a row of a table stored in a database, whilst if a lock has coarse granularity, each lock could correspond to an entire database inside a RDBMS [26][27].

Given the nature of how RDBMS and the solutions that utilize them work, fine granularity is normally the best option for development, given that, as mentioned previously, each lock normally contains data of a single row inside a table, and as such data becomes much easier to organize in the context of a solution by receiving this type of locks sequentially [26][27].

Multiversion Concurrency Control

Whenever a user is altering a certain row of a table inside a relational database management system, that row gets locked from being read, which can be problem whenever multiple reading and writing processes are needed simultaneously.

In order to fix this issue, "Multiversion Concurrency Control" (also referred to as MVCC) is used. Using a process called "Snap Isolation", instead of locking the row that's being worked on and whenever a transaction is requested simultaneously, a previous version ("Snapshot") of that row will be supplied to any other transaction which may run concurrently with the first [28][29]. Utilizing RDBMS that include MVCC as a base for software development regarding data storage minimizes the possible errors that can occur if multiple users utilize the solution to be developed in tandem, whilst they access the same databases for similar purposes.

Relational Database Management System Feature Comparison

A comparison will now be made between features presented by the valid RDBMS, in the context of the problem statement introduced previously. Once the comparison has been made, the best relational database manage-

ment system given the context will be selected.

The comparison will be based on the two previously explained features: "Lock Granularity" and "Multiversion Concurrency Control". This comparison can be seen in Table 2.2.

It is important to note that, since the study of these features is meant to lay the groundwork for the selection of a relational database management system to be used for development of a solution later down the line, the comparison will be limited to the best options for development, namely the RDBMS which either don't have a license or are covered by GPL. Given the context given for the situation surrounding "MySQL", it will also be omitted from the comparison in Table 2.2.

RDBMS	**Fine Granularity Compatibility (Row Level Locks)**	**Multiversion Concurrency Control**
CUBRID	Yes[30]	Yes[30][31]
INGRES	Yes[32]	Yes[33]
LucidB	No (column level locks)[34]	Yes[35]
MariaDB	Yes[36]	Yes[37]
sqLite	No (database level locks)[38]	No (Write-Ahead Logging)[39][40]

Table 2.2: Relational Database Management Systems Comparison (Development Features).

Analyzing the provided Table, most of the mentioned RDBMS have similar features and as such, between "CUBRID", "INGRES" and "MariaDB" in particular, the RDBMS chosen for database development solutions is tied mostly to the personal preference of the developer in the given context.

2.1.2 Market Data Storage Systems

In this sub section, some examples of the existing data storage solutions will be detailed. This includes the features that are unique to each one, as well as the pros and cons of their usage.

Ninox

The "Ninox" data storage solution is divided in two main components: "Ninox Apps" and "Ninox Cloud". "Ninox Cloud" is a subscription based system that utilizes a custom built cloud for database storage. It provides its own security measures plus an user-friendly interface for database manipulation through built-in templates, custom actions, scripting and drag and drop formulas. In spite of having a custom built cloud storage system, it provides its *iOS* users the option of storing their database information on their own "iCloud", providing no such option for *Windows* and *Linux/Android* users. It allows for local storage, however, for any platform [41] [42].

The "Ninox Apps" component of the solution, on the other hand, is an application that provides its users the same user interface provided by the "Ninox Cloud" service, but does not provide a custom cloud service, only allowing for "iCloud" or local storage. This is an *iOS* exclusive app [41] [42].

The "Ninox Cloud" interface is represented in Figure 2.1.

Some of the advantages of using the solutions presented by "Ninox" for data storage are as follows:

- Compact and fully customizable user interface;
- Robust workflow management tools.

The solutions presented by "Ninox" also have some perceivable issues, however:

- Limited features for users of non-*iOS* platforms;
- All data must be manually inserted.

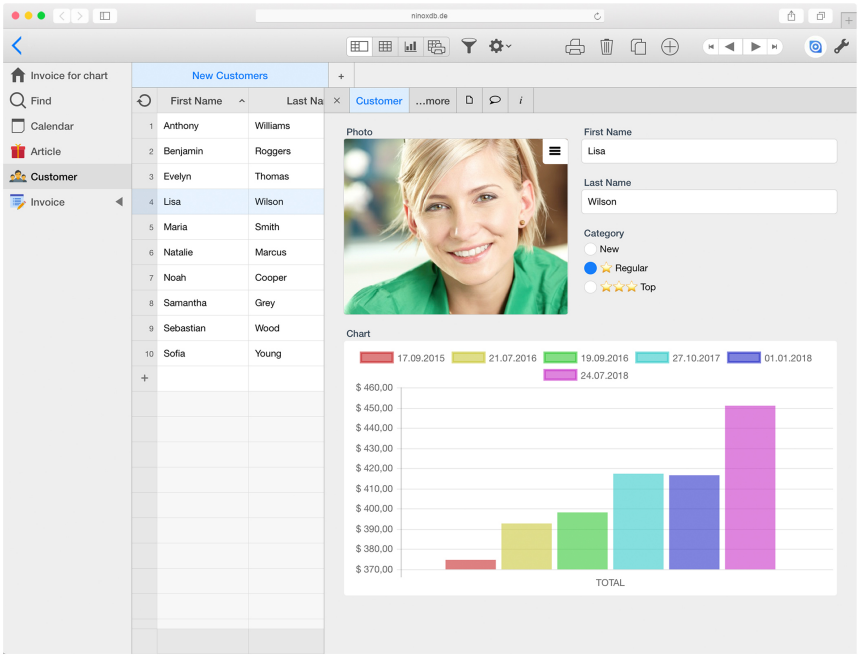


Figure 2.1: "Ninox Cloud" Interface [2].

TeamDesk

The "TeamDesk" solution, developed by a company of the same name, provides cloud-based storage services. The solution presents its users with the ability to create and manage their own databases through a simple but effective interface. The cloud in which the databases are stored is a part of "TeamDesk" itself, providing its users with all of their database needs in one program.

The solutions main draw consists in the "no coding knowledge necessary" approach to program design, as well as its reliability, having a 99.6% service up time in the past 6 years [3].

The solution is, however, a fairly expensive subscription service, whose price changes depending on the amount of users that will be utilizing the solution. The lowest price subscription includes unlimited access up to a maximum of five users, meaning that it is a solution more appropriate for collective usage inside a company.

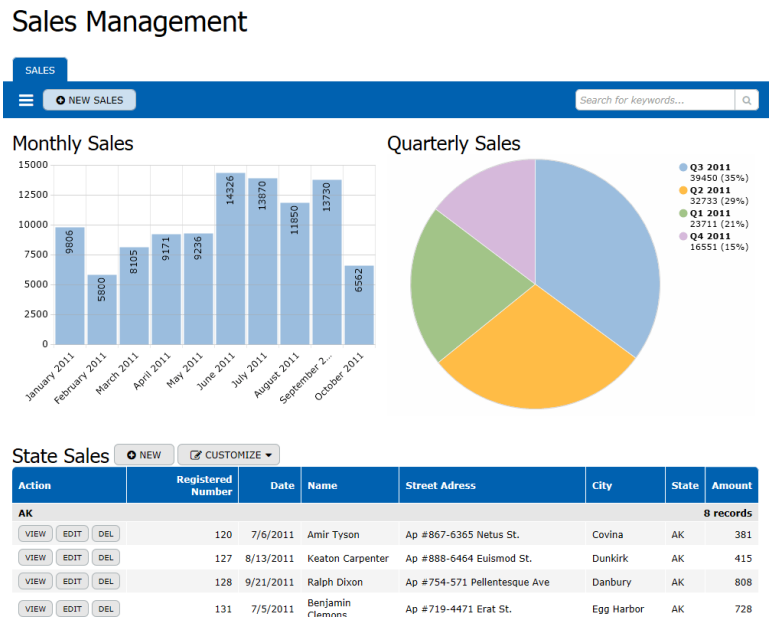


Figure 2.2: "TeamDesk" Data Management Interface [3].

The "TeamDesk" data management interface is represented in Figure 2.2.

The main features "TeamDesk" presents its users are as follows:

- Multiple visual database template options to facilitate workflow;
- Robust architecture, almost no program downtime historically.

The problems of the "TeamDesk" solution, on the other hand, are the following:

- Fairly expensive monthly subscription system;
- All data must be inserted manually.

CASPIO

The "CASPIO" platform is a solution that provides its users with a visual cloud application builder and launcher [4]. This platform's interface is represented in Figure 2.3.

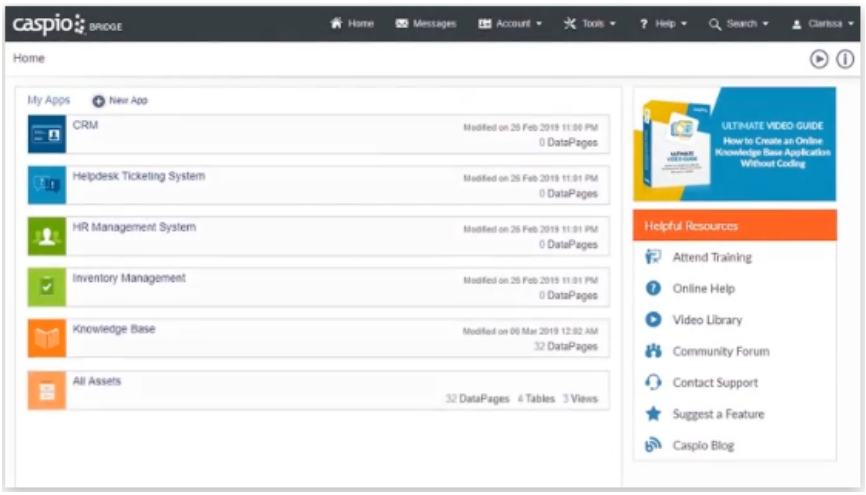


Figure 2.3: "CASPIO" Platform Interface [4].

Similarly to other market data storage systems, it publicizes itself as a "no coding knowledge necessary" solution to the data storage and visualization problem. What sets this solution apart from its counter parts is the simple-to-use block programming features for cloud application development, as seen in Figure 2.4. This feature gives its users the ability to customize their own applications, in both visual and practical terms [4][43].

Even though this solution provides its users relatively more freedom than its competition, that freedom is still confined to the possibilities presented by block programming, which are very specific to their use case compared to conventional programming tools.

Some of the features that "CASPIO" presents are as follows:

- Robust and unique customization tools through block programming;
- Offers an unlimited-time free to use option, unlike most other data storage systems available in the market.

The "CASPIO" solution also has some issues, such as:

- Even though it is provided, the free version of the tool cannot be customized in any way;

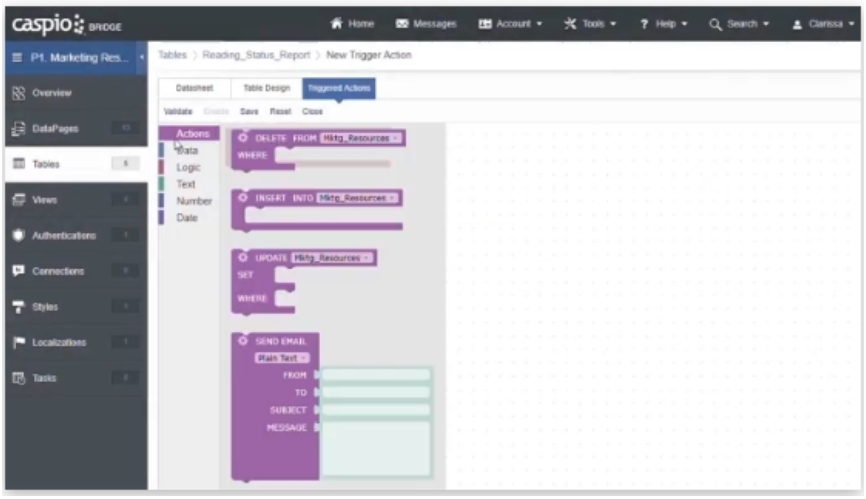


Figure 2.4: "CASPIO" Block Programming Interface [4].

- Fairly expensive monthly subscription for all other versions, relative to other data storage systems.

SolarWinds Backup

The "SolarWinds Backup" data storage solution, developed by "SolarWinds Msp", provides its users with extremely varied cloud data backup and robust data recovery options. This solution not only includes its own cloud system, as it also allows its users to store their data in a plethora of other cloud platforms such as "Office 365 exchange" and "OneDrive", for example.

This solution provides recovery options regardless of the cloud the users decided to store their data into, allowing for cloud-to-cloud data transfers as well as for the traditional data transfer options. Finally, the described solution also optimizes the sent and received data in between storage locales, allowing its users to only send or receive the latest changes being made to the stored data [44].

The main features of the "SolarWinds Backup" are the following:

- Greatest diversity of options when it comes to data storage between all the mentioned data storage systems;

- Robust and efficient data backup systems.

The main problems with the use of the "SolarWinds Backup" are the following:

- The only free option provided to use this solution is based upon a timed trial system;
- Mostly focused on data backup solutions, lacks customization features present in other solutions of the same type.

2.2 Data Visualization Web Tools

In this section, some of the online data visualization tools available in the market will be mentioned. These solutions are meant to be used purely in the context of data visualization, whilst the ones detailed in the previous section are centered around data storage, despite some of them having simple data visualization options. Also, the solutions that will be described in this section are mainly accessible through browsers, in order to facilitate data visualization access to as many interested parties as possible.

These include a Robot Operating System (ROS) web integration, along with more conventional web visualization solutions.

2.2.1 Grafana

The "Grafana" solution was created with the intention of providing a platform capable of analyzing all the metrics that any user could have. This solution allows its users to query, visualize and interpret metrics, regardless of which database management system they are stored in, providing customizable dashboard displays for data progression observation and interpretation [5].

In Figure 2.5, an example of these dashboards is represented.

Figure 2.5: "Grafana" Dashboard Example [5].

The fundamental features provided to a user when using the "Grafana" solution are as follows [5]:

- Quick and efficient visualization of data through dynamic dashboard displays;
- Easy-to-use visual dashboard development system;
- Embedded data acquisition from a plethora of different RDBMS;
- Customizable data threshold alerts;
- Fully open-source, with facilitated support for community developed plugins.

The mentioned visual dashboard development system allows users to customize their dashboards by dragging and dropping desired elements into a dashboard, elements which can include plugins developed by the "Grafana"' community if needs be.

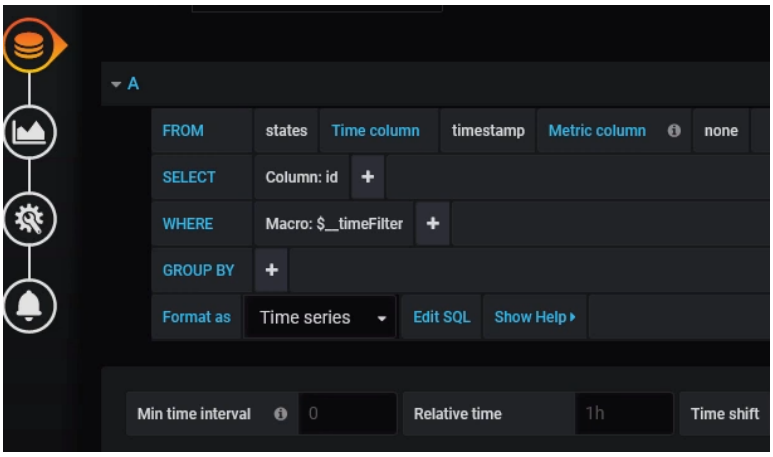


Figure 2.6: "Grafana" Query Editor for "MySQL" [6].

Be that as it may and even though the development of the mentioned dashboards is relatively simple through the use of this tool, it still requires that the user creating the dashboards has a decent grasp on the basic configuration of its preferred RDBMS and the data that exists within it.

On top of that, basic understanding of the queries required to obtain data from the preferred RDBMS is necessary, as can be seen by the way the data acquisition is setup for a graph in Figure 2.6, which is applied to the "MySQL" RDBMS. This can be a major issue for users who simply want to visualize stored data in databases and have no experience whatsoever with RDBMS queries.

2.2.2 Webviz

The "Webviz" solution consists in a ROS web visualization tool developed by "Cruise". This solution was created with the intent of facilitating the ROS-based development of self driving cars. [45].

The main concept behind this web visualization tool was originally to improve accessibility of test data at any point for any user that was given access to that data. As such, a website was created that allowed its users to drag and drop ROS-logged information for any test (also known as "ROS

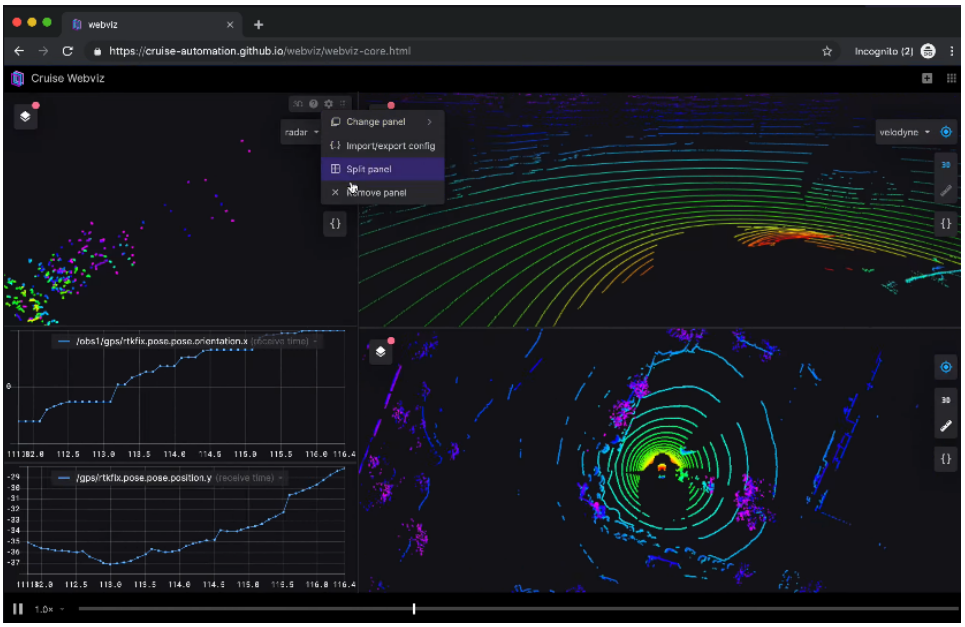


Figure 2.7: "Webviz" Layout Example [7].

bag") into the website itself. Once the file was dropped into the website, the "ROS bag" would be parsed and all the points logged inside that file would be shown to the user in a tri-dimensional environment in a customizable layout, as shown in Figure 2.7. This customized layout would then be accessible by anyone allowed to use the website at the time [7][45].

This solution was made public after the development of the desired self-driving technologies by the company was concluded. However, given that this tool was made for a very specific purpose that is the development of self-driving car technologies through ROS, the amount of supported web browsers that can run it is limited to one, namely "Google Chrome", since it was the browser used by the responsible company during development [7][45]. Besides this, the web solution is only capable of parsing data logged in the aforementioned "ROS bags", which adds another level of specificity to the overall process.

2.2.3 European Environment Agency's Interactive Data Set Viewers

The European Environment Agency (EEA) is an agency of the European Union, which has the objective of supporting the sustainable development of the countries that comprise it. This is done through the provision of timely, targeted, relevant and reliable information to policy making agents and the public, regarding sustainability [46].

In order to achieve the goal the EEA was set out to achieve, data all across European countries was acquired and presented to the public in the form of interactive data viewers, separated by characteristics. Each of these data set viewers contain different types of graphs depending on the characteristics they are associated with, some of them even including dashboards in which all the graph data existing for the observed characteristic can be interpreted at a glance. In Figure 2.8, some of the aforementioned data set viewers are shown. On top of that, in Figure 2.9 an example of a data set viewer is represented, more specifically one relating to the change in urban waste water treatment in European countries, from 1990 to 2017.

Given that the acquired characteristics were taken from all over Europe, there is a huge amount of data set viewers available through the EEA interactive data set viewer page, showcasing a total of 1239 different interactive data set viewers at the time of writing this thesis [8].

In order to improve filtering between all these different data set viewers and facilitate the search for the right information given an users' needs, each of the data set viewers are grouped into categories based on the data they represent. These groups are defined by categories ranging from "Agriculture", which includes notable geographical and chemical data sets in the context of agriculture in Europe, to "Air pollution", which includes data sets regarding the chemical characteristics of the air in different areas of Europe relevant to the study of air pollution.

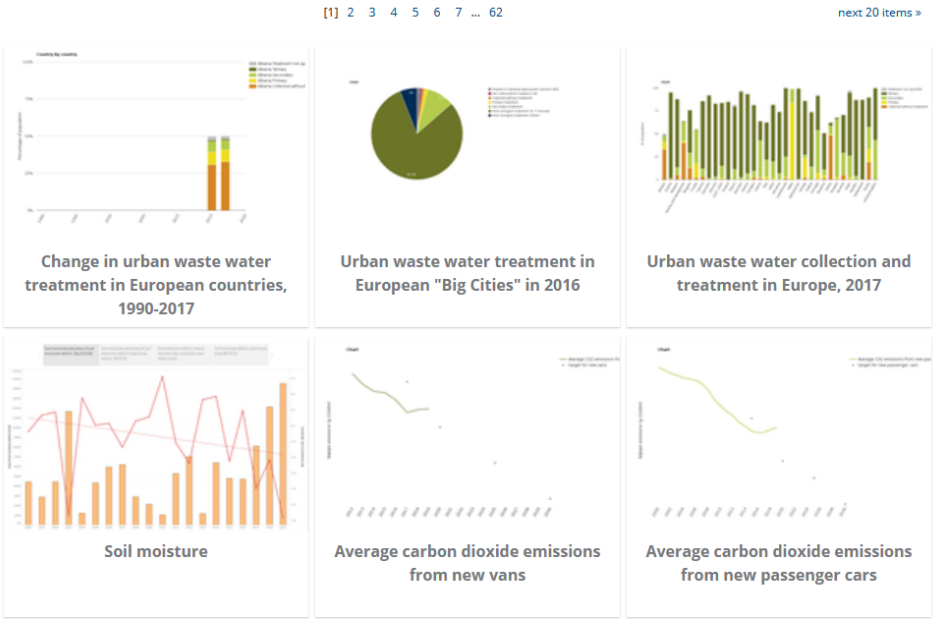


Figure 2.8: EEA Data Set Viewer Options Example [8].

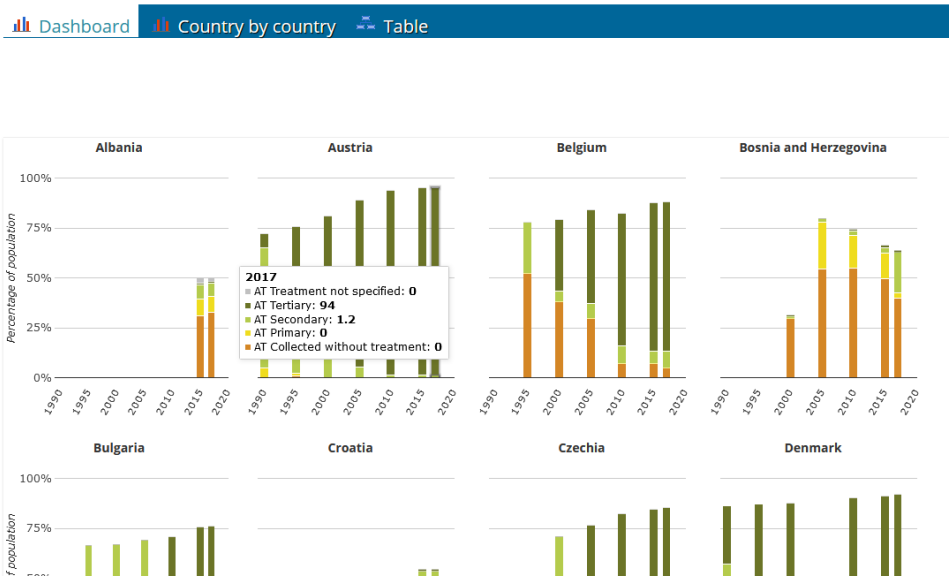


Figure 2.9: EEA Data Set Viewer Example [8].

It is important to note that only one category can be chosen at a time in regards to filtering, but some of the data set viewers in these categories can overlap. For example, besides the "Agriculture" category there is a "Soil" category, which includes data sets regarding multiple observed characteristics of the soil across Europe, making it so data set viewers regarding soil moisture can be and are shown in both of them [8].

Since all of these interactive data set viewers were published and can be updated at different times, the EEA's interactive data set viewer page also allows for filtering by the time frame between the first time a data set viewer was published and the last time it was updated, in tandem with the previously described filtering by category.

As an online data viewer solution though, these interactive data set viewers are only applicable to data stored in the EEA's databases, and need to be specifically tailored for any new data set that is added to those databases, if no existent data set viewer is applicable to it.

2.2.4 ArcGIS Online

The "ArcGIS Online" solution is a tool set for online geographical data visualization development, developed by "Esri". Despite not being an online visualization tool itself, it facilitates the creation of easy-to-access browser-based data visualization solutions, based around geographical data representations [47].

The tool set allows its users to create and share applications created based upon the maps provided, allowing for analysis of the provided data sets in the preferred context. In order to facilitate development even further, there are add-ons provided by the tool set that allow for direct integration of Internet of Things (IoT) interfaces into the web visualization itself, fetching data sets directly and dynamically as they're available.

An example of a data viewer developed using "ArcGIS Online" is the

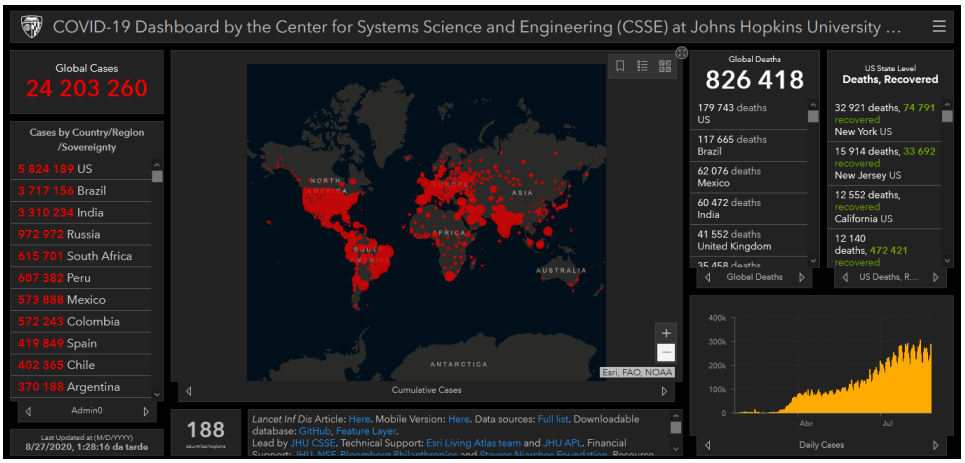


Figure 2.10: Example of a Map Visualization Developed Through "ArcGIS Online" [9].

"COVID-19 Dashboard" developed by the Center for Systems Science and Engineering (CSSE) at John Hopkins University (JHU), represented in Figure 2.10.

This data viewer consists in a dashboard that combines all the known geographical data regarding COVID-19 and its prevalence around the world. This is conveyed through a geographical hot-spot representation in a world map combined with a numerical study relating to either the world or a specific country, which observes how many people are infected, how many people have recovered and the number of deaths caused by the disease [9].

2.3 Developmental Coding Language Options

Having established some of the available market solutions for database management and the browser-based data visualization options, it is also important to determine which coding languages can be used to develop an original database manipulation tool and data visualization web tool.

Initially, some of the developmental coding language options more suited for database management tool development will be detailed, being then fol-

lowed by the coding languages that can be used to develop a data visualization web tool.

2.3.1 Database Manipulation Tool Development

C

C is a general-purpose programming language that was historically used to develop anything from operating systems to even interpreters for other programming languages, such as *Python* [48].

In the context of the development of a database management tool, the *C* coding language is capable of developing solutions that can connect to and manage data relating to tables inside databases in RDBMS [49].

Lua

Lua is an efficient and lightweight coding language that supports procedural programming, object-oriented programming, functional programming, data-driven programming, and data description [50].

When it comes to the development of a database management tool, this language includes the fundamental features to establish a connection to RDBMS and, consequentially, develop scripts that can acquire data from and write data into tables inside a database [51].

Python

Python is an easy-to-use open-source programming language that boasts a considerable amount of community developed libraries, on top of an already robust standard library, that provide a large amount of flexibility when developing solutions [52].

Similarly to the *C* and *Lua* programming languages, this language can setup a connection to tables inside a database in RDBMS and alter them,

being then a viable option for the development of a database management tool [53].

Summary

By analyzing the provided programming languages, we can see that every single one of them allows for the development of solutions capable of managing data inside RDBMS on its own and, as such, the choice of programming language to be used boils down to developer preference.

2.3.2 Data Visualization Web Tool Development

Hypertext Markup Language (HTML)

The Hypertext Markup Language (HTML) is the building block of the web, defining the overall structure of its content [54].

In the context of the development of a data visualization web tool HTML is essential, given that without it the resulting web solution would have no structure and, consequentially, would not work.

PHP: Hypertext Preprocessor (PHP)

The PHP: Hypertext Preprocessor (PHP) is an open-source general-purpose scripting language that is generally suited for web development and can be embedded into HTML. This language requires a web server to run and is interpreted in it, which makes it so its scripts are ran before the HTML is loaded, meaning that all the PHP script outputs will be considered part of the web page structure defined by the HTML [55].

When developing a data visualization web tool, the PHP scripting language is capable of making certain structural elements of a web page conditional and is even capable of interacting with data stored inside databases through its scripts. This makes the PHP scripting language an important

part of a data visualization web tool's development, but it is in no way as essential as the HTML.

Cascading Style Sheets (CSS)

The Cascading Style Sheets (CSS) is a style sheet language used to describe how elements of the HTML structure will look. This style sheet language is ran on the side of the client, more specifically it is interpreted only after the HTML is loaded [56].

Its importance for data visualization web tool development is centered around how the web page itself looks, which makes it important to be used visually, but not as much in a practical sense.

JavaScript

Lastly, *JavaScript* is a compiled programming language for web pages. Like CSS, it runs client-side and is used to design/program how the web pages behave on the occurrence of an event [57].

One key difference between CSS and *JavaScript*, along with the reason for the importance of *JavaScript* when developing a web data visualization tool, is that *JavaScript* is dynamic, being capable of defining responsive graphs and charts, whilst CSS is fixed, being tied to characteristics like object color and allocated space inside a page. Taking this into account, the importance of the described programming language is that of dynamic data displays and a browser-based data visualization tool wouldn't make sense without them.

Summary

Summarizing the described web development languages, all of them can and should work in tandem to achieve a proper data visualization solution, despite CSS being mostly used for visual purposes rather than functional.

```
Press  Temp   Cond   Sal    O_O2%   O_O2ppm  pH    Chl(a) Tur(FTU) PThrin Time&Memory
   5.89 16.934 44.794 34.968  102.59    8.02  8.211   0.11   0.69  -0.23 23:32:16.55M+
   5.93 16.934 44.798 34.971  102.55    8.02  8.211   0.34   0.60  -0.05 23:32:17.50M+
   5.90 16.948 44.812 34.972  102.55    8.02  8.211   0.34   0.67   0.18 23:32:18.61M+
   5.90 16.974 44.839 34.973  102.59    8.03  8.211   0.33   0.65  -0.09 23:32:19.56M+
   6.00 16.967 44.827 34.968  102.59    8.03  8.211   0.43   0.63   0.04 23:32:20.67M+
   6.04 16.956 44.818 34.969  102.71    8.03  8.211   0.47   0.63  -0.08 23:32:21.62M+
   5.91 16.935 44.792 34.965  102.71    8.03  8.211   0.36   0.80  -0.14 23:32:22.73M+
   5.83 16.926 44.789 34.970  102.58    8.02  8.211   0.61   0.70  -0.10 23:32:23.68M+
   5.85 16.927 44.790 34.971  102.58    8.02  8.211   0.39   0.71  -0.00 23:32:24.79M+
   5.84 16.932 44.801 34.975  102.65    8.03  8.211   0.21   0.71  -0.11 23:32:25.74M+
   5.78 16.973 44.848 34.981  102.65    8.03  8.211   0.52   0.65  -0.14 23:32:26.84M+
```

Figure 2.11: Example of System Sensor Data Stored in a Text File ("OS311" Multi-parameter pH Probe).

2.4 Data Output Scenarios

Lastly, the different ways that sensors can output their data in a general sense will be described.

If the system the sensors are attached to does not have any way to connect to a system capable of storing and/or parsing the data directly, a storage component should exist within the system itself so that visual data can be saved and the written data can be kept in text files for later parsing, once the system can be accessed. An example of a text file acquired through this method can be seen in Figure 2.11, taken from the "OS311" multi-parameter pH probe [58], which outputs a line relating to the header of each column along with the numerical data retrieved from its sensors every few cycles of data logging.

The Comma-Separated Values (CSV) file, in which the all of the characteristics acquired by a sensor or group of sensors are output into a text file separated by commas, is another example of how data can be output by sensors or groups of sensors into text files.

A good example of a sensor that does this is the "SWS-050" visibility sensor [59], as can be seen by its output in Figure 2.12.

On the other hand, if the system has some way of direct connection to an outside system capable of storing and/or parsing its data, be it wired or

```
01\03\20 00:00:03,SWS050,001,060,40000 M,00,000.07,XOOj
01\03\20,00:01:03,SWS050,001,060,40000 M,00,000.07,XOOk
01\03\20,00:02:03,SWS050,001,060,40000 M,00,000.07,XOOl
01\03\20,00:03:03,SWS050,001,060,40000 M,00,000.07,XOOm
01\03\20,00:04:03,SWS050,001,060,40000 M,00,000.07,XOOn
01\03\20,00:05:03,SWS050,001,060,40000 M,00,000.07,XOOo
01\03\20,00:06:03,SWS050,001,060,40000 M,00,000.07,XOOp
01\03\20,00:07:03,SWS050,001,060,40000 M,00,000.07,XOOq
01\03\20,00:08:03,SWS050,001,060,40000 M,00,000.07,XOOr
01\03\20,00:09:03,SWS050,001,060,40000 M,00,000.07,XOOs
01\03\20,00:10:03,SWS050,001,060,40000 M,00,000.07,XOOk
01\03\20,00:11:03,SWS050,001,060,40000 M,00,000.07,XOOl
01\03\20,00:12:03,SWS050,001,060,40000 M,00,000.07,XOOm
01\03\20,00:13:03,SWS050,001,060,40000 M,00,000.07,XOOn
01\03\20,00:14:03,SWS050,001,060,40000 M,00,000.07,XOOo
01\03\20,00:15:03,SWS050,001,060,40000 M,00,000.07,XOOp
```

Figure 2.12: Example of System Sensor Data Stored in a CSV File ("SWS-050" Visibility Sensor).

wireless, that data can be sent in real-time directly to the outside system through socket communication or it can also be sent progressively through text files, like the ones shown, over certain periods of time through wireless connection.

Chapter 3

Requirement Analysis

In this chapter, the concepts that lay the foundation for the solution that is the object of this thesis will be described, that being the aforementioned database storage and web visualization tools. These concepts can be divided into three main sections: sensor data characteristics, sensor data output of autonomous systems and web visualization.

The sensor data characteristics section consists in a theoretical study of what types of output can be expected out of sensors associated with any autonomous system.

The section relating to the sensor data output of autonomous systems incorporates the different sensors an autonomous systems may have, as well as the ways the data output from those sensors can be grouped to facilitate decision making.

The web visualization section, on the other hand, consists in the basics behind every good online data visualization tool. This section also describes why the use of this type of tool is advantageous for the solution at hand.

3.1 Sensor Data Characteristics

This section is dedicated to exploring the different types of output expected from autonomous systems and their sensors. The importance of this is due to the fact that the desired solution will have to be as generally applicable as possible and, by analyzing autonomous systems and their sensors, proper conclusions can be drawn in regards to the features necessary to accommodate as many different autonomous systems in the desired solution as possible.

3.1.1 Essential Vs. Formative Sensor Data

Before tackling the data output of different autonomous systems, there should first be a clarification of what essential and formative sensors are in the context of this thesis, and their importance in this section.

Essential sensors of an autonomous systems are fundamentally those that allow it to function and are used exclusively for that purpose. A good example of this are the gyroscope and accelerometers attached to self-driving cars. The data acquired from these sensors isn't necessarily output, however it is crucial to parse in order to employ proper control over the car regardless of the environment it finds itself in.

Formative sensors of an autonomous system, on the other hand, are sensors that are not necessary in anyway for proper autonomy of a system, but instead provide data relating to the environment the autonomous system that uses them finds itself in, in a way that it can be output for human observation. An example of this are the oxygen level sensors attached to an autonomous system tracking sea floor data. The data acquired from this sensor has no use for the automation of the system that includes it, but it is of great interest when output to an user, in order to draw conclusions about the area that the autonomous system finds itself in.

When it comes to the sensor data outputs to be described over this

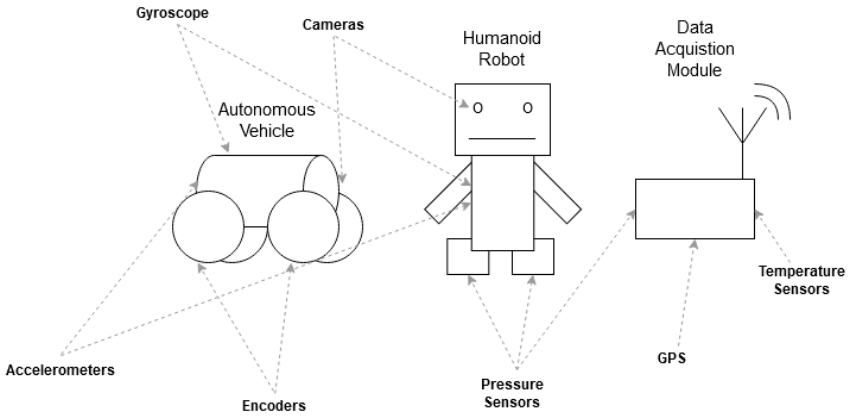


Figure 3.1: General Autonomous System Sensors.

section, they will be mostly focused around the aforementioned formative sensors. This is due to the context provided by the original problem statement and objective of this thesis, which is the development of a system that facilitates storage of and decision making based on sensor data, in regards to environmental observation.

3.1.2 General Sensor Data Output

Before describing practical cases of sensor data output, it is first necessary to understand what types of sensors are to be expected from different autonomous systems.

To do this, a general demonstration of the possible sensors autonomous systems can have is shown in Figure 3.1.

Analyzing Figure 3.1, a representation of an autonomous vehicle, humanoid robot and a data acquisition module can be seen (from left to right, respectively). Each one of these autonomous systems include both essential sensors, such as the accelerometers and gyroscopes, and formative sensors, such as the temperature sensors and the Global Positioning System (GPS).

It is important to note that, depending on what autonomous system they are paired with, different sensors can have different purposes even to

the point of changing their fundamental importance in the context of an autonomous system.

A good example of this is the case of the cameras. In the context of a humanoid robot, the cameras can be considered formative sensors, whenever they are used in the context of simple visual data acquisition, without the need for any actual processing. This stems from the fact that the data is acquired for no other reason than that to be visualized by the end user, not to make possible essential processes conducted by the autonomous system in question.

When applied to an autonomous vehicle, on the other hand, the cameras can become essential, in the context of environment traversal. The reason behind this is that the cameras can be used to detect obstacles in the autonomous vehicle's path, making the cameras essential for its autonomous movement throughout different environments.

3.2 Sensor Data Output of Autonomous Systems

In this section, the study of the different sensors included in a plethora of different autonomous systems will be conducted, along with their associated data output. By observing and detailing the outputs of all the sensors inside a variety of different autonomous systems, it is possible to infer all the necessary requirements for a generally applicable data visualization tool.

The general study will focus 3 different autonomous systems, namely the "MarinEYE", "FeedFirst" and "TowFish". In the case of the "TowFish", some additional sensor output regarding the project that this autonomous system finds itself in will also be described.

The study was confined to these autonomous systems for the sake of brevity, given that there are always going to be differences between the way autonomous systems output their data and, as such, the similarities that they can share are what needs to be taken into account.

3.2.1 MarinEYE

The "MarinEYE" is an autonomous system used to monitor biological characteristics in a marine environment, providing important information when it comes to responding to time sensitive ocean issues and evaluating the environmental status of marine ecosystems. This system also aims to tackle the issue relating to the lack of readily available oceanic data, in order assist with public and private sectors connected to the sea.

Some application examples for the "MarinEYE" are weather studies, resolution of marine transportation accidents and pollution and decision making relating to the marine energy and aquaculture sectors [60].

The "MarinEYE" combines a multi-parameter sensor suite, water filtration fraction system for Ribonucleic Acid (RNA)/ Deoxyribonucleic Acid (DNA) sampling, plankton imaging system, environmental acoustics logging technologies and a GPS combined with a wireless communication link.

This is done through a compact, modular system interface that can be coupled with both fixed and mobile platforms [60].

The described "MarinEYE" autonomous system is represented in Figure 3.2.

Multi-Parameter Sensor Suite

The multi-parameter sensor suite obtains physical and chemical information of the underwater environment it may find itself in.

This sensor suite includes Conductivity, Temperature and Depth (CTD) sensors, along with oxygen, fluorescence and turbidity sensors [60].

RNA/DNA Sampling

The sampling system developed for the "MarinEYE" collects multiple different-sized planktonic samples and stores them in a preservative solution, so

Figure 3.2: MarinEYE. Taken From [10].

that they can be kept for a long period of time, being used for genomic (DNA/RNA) analysis [60].

Plankton Imaging System

The imaging system present in the "MarinEYE" is one of its main feature, and has 2 main objectives:

- Collecting high resolution images of the planktonic organisms with various different group sizes;

- Providing information suitable for estimating the overall abundance of particular groups of plankton in the area.

Besides plankton imaging, this system also allows for the acquisition of standard digital images of its surrounding environment. These images can be used to increase awareness and/or take snapshots of marine specimens, such as fish or marine mammals [60].

Acoustic System

The acoustic system embedded in the "MarinEYE", just like the imaging system, has 2 main purposes:

- Marine sound recording - Recording acoustic noise, focused on biologically generated signals, such as those from marine mammals;
- Acoustic assessment - Assess the presence of schools of fish, their number and size in the water column, along with the estimation of their biomass.

It's important to note that these 2 objectives are performed through 2 different subsystems [1]. The marine sound recording is done through an hydrophone, covering frequencies between 10 Hz and 200 KHz, with appropriate low noise signal conditioning and data acquisition.

On the other hand, the acoustic assessment is performed with a dual frequency (90/300 KHz) echo sounder with 50 m of range [60].

GPS and Wireless Communication Link

Lastly, the "MarinEYE" includes a GPS, with the intent of locating all the data acquired by the previous groups of sensors in time and geographical space.

As for the wireless communication link, although not useful underwater, it facilitates data transfer and communication with outside systems when the "MarinEYE" is at surface [60].

Optimal Sensor Data Grouping

When it comes to the sensor data that should be supplied to a storage and web visualization tool, in order to achieve the right conclusions in the context of the "MarinEYE", it needs to be grouped depending on the purpose of each sensor.

The sensor data pairing should be done between sensors with the same purpose and goal in the context of the "MarinEYE", namely the acoustic system, plankton imaging system and RNA/DNA sampling should all be correlated in order to establish, beyond reasonable doubt, which animals exist in a certain area.

The multi-parameter sensor suite, on the other hand, should be separated from the previous group, given that it is used to make assessments of the underwater environment, foreboding the animals within it.

Both these groups need to be paired with the geographical information acquired from the integrated GPS, in order to properly understand the areas in which the autonomous system was used.

3.2.2 FeedFirst

The "FeedFirst" autonomous system was created with the goal of improving the efficiency of the exogenous feeding of marine fish larvae kept in aquaculture, in order to achieve a better developmental stage for these larvae. Due to the fact that most commercial microdiets are poorly accepted, ingested and digested by the larvae given their sub optimal nutritional composition, the introduction of an nutritionally balanced inert diet is normally only possible after a few weeks of larval development [17].

The "FeedFirst" solution aims to solve this by introducing high-quality inert microdiets and a rearing tank that provides innovative hydraulic features. The tanks allow for longer residence time of diets in the water column, giving the larvae more time to consume those diets, whilst also incorporating self-cleaning mechanisms to achieve better sanitary condition in the rearing water [17].

The solution includes sensors that output data regarding water temperature, pH, dissolved oxygen, turbidity and ammonia levels. The solution also incorporates a camera, which retrieves visual information from the interior

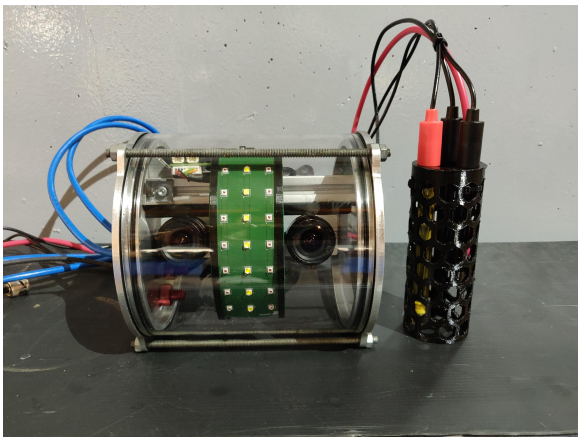


Figure 3.3: “FeedFirst”.

of the tank and, through image recognition software, highlights the more important observable features, such as the presence of plankton, inside the tank. The “FeedFirst” autonomous system is represented in Figure 3.3.

It is important to note that an online visualization tool was created for this project, in order to better track the data from all the previously described sensors. In Figure 3.4 a screenshot of the data visualization tool is shown. This screenshot was taken during a testing phase of the “FeedFirst” solution, so the design of the page and the data shown may not be final.

Optimal Sensor Data Grouping

Since the “FeedFirst” project’s main goal revolves around the analysis of the characteristics of water, in order to determine its quality, all the data relating to these characteristics should be grouped with one another. This includes the sensor data relating to the water temperature, pH, dissolved oxygen, turbidity and ammonia levels.

As for the data output of the cameras, it does not necessarily need to be grouped with other sensors, given that these cameras were meant to function as visual feedback in regards to the inside of the tank.

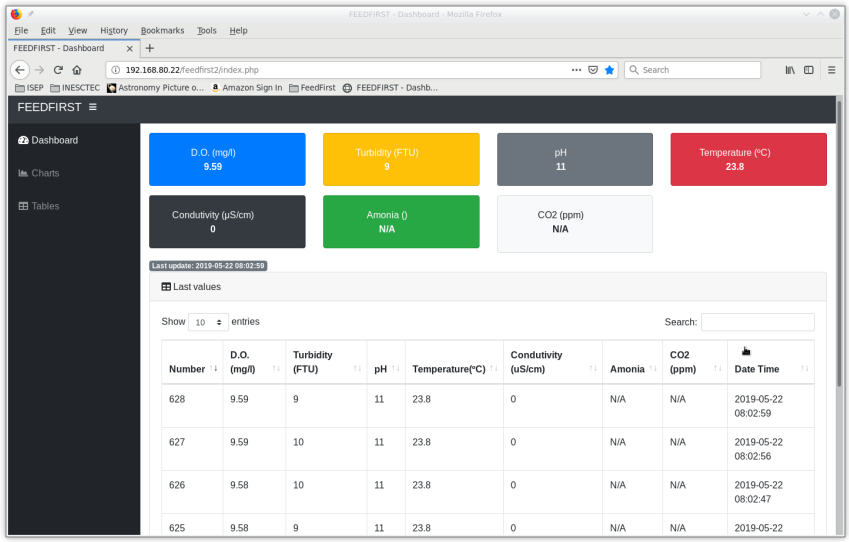


Figure 3.4: "FeedFirst" Data Visualization Tool.

3.2.3 SAIL

The "SAIL" project, developed by investigators and teachers from *Instituto Superior de Engenharia do Porto* (ISEP) and *Instituto de Engenharia de Sistemas e Computadores* (INESC TEC), has been conducted in order to establish a new measurement of the planet's atmospheric electric field and collect information that hasn't been updated for the past 100 years [16].

To do so, a series of sensors along with an autonomous system (that contains its own group of sensors) were installed in the school-vessel *Sagres*, during its expedition around the globe. The mentioned school vessel is shown in Figure 3.5, along with its electric field sensors, whilst they were being calibrated. These electric field sensors and their utility will be described later in this sub section. The itinerary for the previously mentioned trip around the globe is represented in Figure 3.6.

To do this, the two electric field sensors shown in Figure 3.5 were installed in one of the school-vessel's masts and, along with that, the vessel was fitted with a visibility sensor, ion counter, gamma ray sensor and GNSS. Adding

Figure 3.5: School-Vessel *Sagres* and Electrical Field Sensors [11].

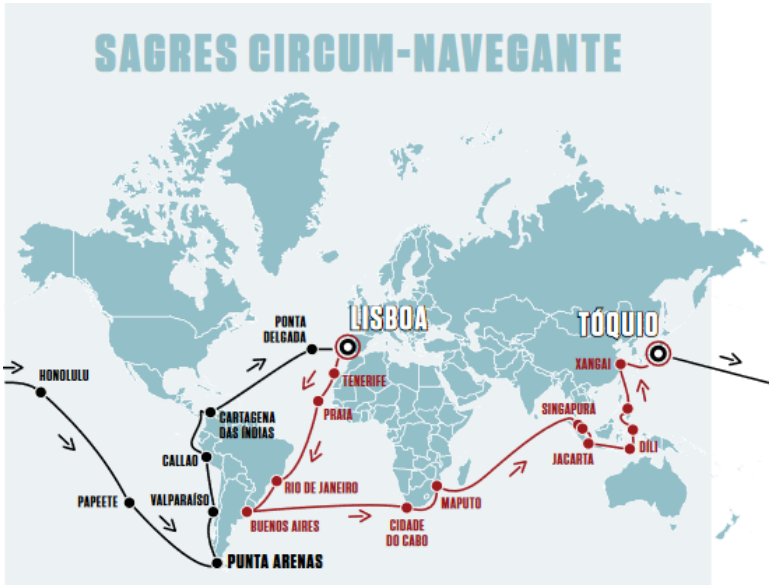


Figure 3.6: "SAIL" Project Itinerary [12].

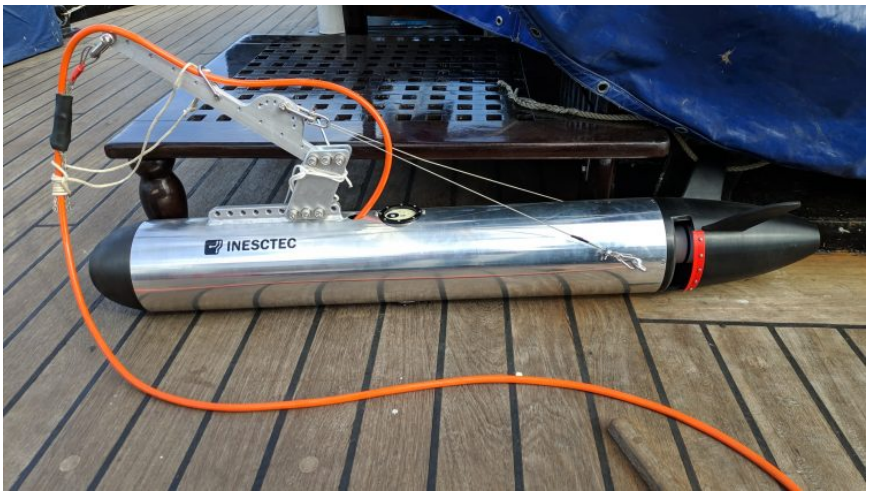


Figure 3.7: "TowFish" [13].

to all of these sensors, the autonomous system "TowFish" was developed with its own set of sensors, with the goal of obtaining underwater data and pairing it with all the above water data taken by the vessel as described [16].

TowFish

The "TowFish" autonomous system was developed for use in the "SAIL" project in order to acquire a plethora of different underwater characteristics through it's set of sensors [16]. This autonomous system includes hydrophonic sensors along with CTD sensors. It also includes chlorophyll and oxygen sensors, so that it can obtain data as a way to assess how healthy its surrounding environment is for the fish.

The described "TowFish" is represented in Figure 3.7.

Optimal Sensor Data Grouping

First and foremost, given that the main purpose of the "SAIL" project is to establish a new measurement of the planet's atmospheric electric field, the data sets that should be grouped and sent to visualization tools are those relating to the electrical fields acquired from the sensors installed in the

school vessel's mast and the geographical positioning data obtained from the school vessel itself.

Once that data has been properly grouped, the secondary goal of acquiring data from the under sea ecosystems with the use of the "TowFish" should be tackled. In order to do so, the geographical data from the school-vessel that was grouped with the electrical field information, should also be grouped with all the sensor data from the "TowFish" itself, so that a more coherent study of different under sea ecosystems can be conducted.

3.2.4 Sensor Data Summary

By observing the sensor data output characteristics of the aforementioned autonomous systems, they are divisible into different groups when it comes to their importance in the context of a project along with how they should be visualized.

The different groups of sensor data are divided as follows, based on their objectives:

- Visual and sound-based spacial perception;
- Environmental characteristic acquisition;
- Geographical positioning.

Visual and Sound-Based Spacial Perception

This group of sensor outputs is tied to both cameras and SONAR systems, and the main goal behind their usage is to improve the understanding of its surrounding area at a physical level.

When it comes to visual spacial perception sensors, they simply provide imagery of the surrounding area that they are pointed towards, and don't require any data parsing in order to draw conclusions from. Observing the images/recordings from visual perception sensors is generally enough for any

user to determine what it needs, although image recognition software could be paired with these sensors to highlight certain visual characteristics from the environment that the user sees fit.

On the other hand, sound-based spacial perception sensors provide multiple measurements between themselves and whichever physical obstacle that exists in the vicinity and, by associating all the acquired measurements with the position of the sensors, a surface can be perceived at a certain distance of the sensors through an agglomerate of points referred to as "point clouds". These "point clouds" could be observed without any posterior processing and still yield fair conclusions, but ideally they should be used to create a tri-dimensional walls inside a simulated environment, in order to facilitate the understanding of the surrounding space perceived by the sensors.

Environmental Characteristic Acquisition

The sensors that are used in the context of environmental characteristic acquisition are those that acquire chemical data of their surrounding area. This data can range from salinity, conductivity and temperature to oxygen and turbidity levels of the environment. The aforementioned data can be shown to the user without much, if any, filtering or post-processing, since it's all self-contained numerical data. One way to draw more conclusions from these sensor's data is to insert them into a graph and observe the variation of values over time in a certain area.

Geographical Positioning

Lastly, the sensors that provide geographical positioning data, such as the GNSS, provide the latitude and longitude of their current position. This data is relatively useless, when it comes to drawing conclusions from in isolation, but can be a powerful tool when paired with data from other groups of sensors. For example, during an expedition over the sea it is

possible to the establish chemical data of a large area by associating the data from chemical sensors over time with geographical coordinates.

3.3 Web Visualization

In this section, the importance of web visualization tools will be described, along with the necessary components for a successful implementation of these tools, in the context of sensor data visualization.

3.3.1 Importance of Using a Web Based Visualization Tool

The use of web-based visualization tools is a great solution to problems relating to data sharing and interpretation between individuals in a group/organization. Without a web-based solution, data is normally shared through cloud services, or even physical Universal Serial Bus (USB) flash drives, in the form of data logs, which then need to be parsed either manually or through the use of programs such as *MATLAB*[1]. The use of a web-based visualization tool makes it so no actual files need to be shared between users, and the data parsing itself can be done through the developed website.

One other major importance of using a web-based visualization tools is the security associated with it. The number of users that have access to any website can be curated and, since there is no direct sharing of files containing confidential information, data leaks are fairly less likely than other traditional methods of data sharing.

3.3.2 Web Visualization Design Requirements

Accessibility

The ease of access to any web visualization tool is one of its greatest strengths. By running a single machine that contains all of the databases, using it to

[1] *MATLAB* is a programming platform centered around data analysis and algorithm development [61].

host a website and having both these components interact with each other, any user that is given a Uniform Resource Locator (URL) to the website can access information stored in databases. How the information is displayed depends then on how the website itself is built.

Simple and Clear Interface

The backbone of any good visualization tool is a simple, clear and functional interface. By providing users with an interface that includes such characteristics, the information they seek becomes much easier and quicker to access, which in turn allows for faster and more educated decision making based upon the information provided.

Concise and Informative Data Display

In order to further facilitate decision making based upon provided data, the way the data is displayed must be as direct and objective as possible. In other words, the users that seek the data available through any web visualization tool should not have to go through secondary hurdles such as interpretation of convoluted graphs in order to draw conclusions from that data.

It is also important that the way the data itself is displayed is the correct one for its use case. If, for example, some of the data being shown relates to geographical positions, it is much more evident what that information conveys if it is presented as points in a world map, rather than simply showing it in a graph or table.

3.3.3 Summary and Prototype of a Web Based Visualization Tool

Based upon the described design requirements, this sub section will summarize the fundamental features of a data visualization tool when applied to

the context of this thesis. In order to better understand that summary, a visual prototype will also be provided in tandem in a case by case basis.

Before summarizing what a successful web visualization tool would look like, it is important to note the relative differences in the data that could be acquired for it, which implies different visualization options.

If the data is acquired and to be visualized in real-time, it generally implies that the visualization options provided should be centered on those that provide immediate status information on each of the characteristics provided by the data, not showing too many (if any at all) of the older data values taken during the data acquisition process.

On the other hand, if the data is meant to be visualized as a whole, regardless on how it is being acquired, the visualization options provided should be able to allow its user to observe the data value progression over time of multiple different variables.

Real-Time Data Visualization Page

When it comes to how a real-time data visualization page should look like, the focus should be around immediate and concise data display of the current context. What this means is that, once an user decides to observe data in real-time, this page could be opened at any time and, at a glance, facilitate the understanding of the current status of every characteristic that is being observed through this page.

Pages whose purpose is to observe data in real-time, regarding the current status of each characteristic provided by it, are called dashboards. These dashboards are meant to provide the maximum amount of information to the user possible in an immediate fashion, and whose concepts should be fundamentally utilized when developing any real-time data visualization page.

With that said, a prototype of a dashboard including all of these char-

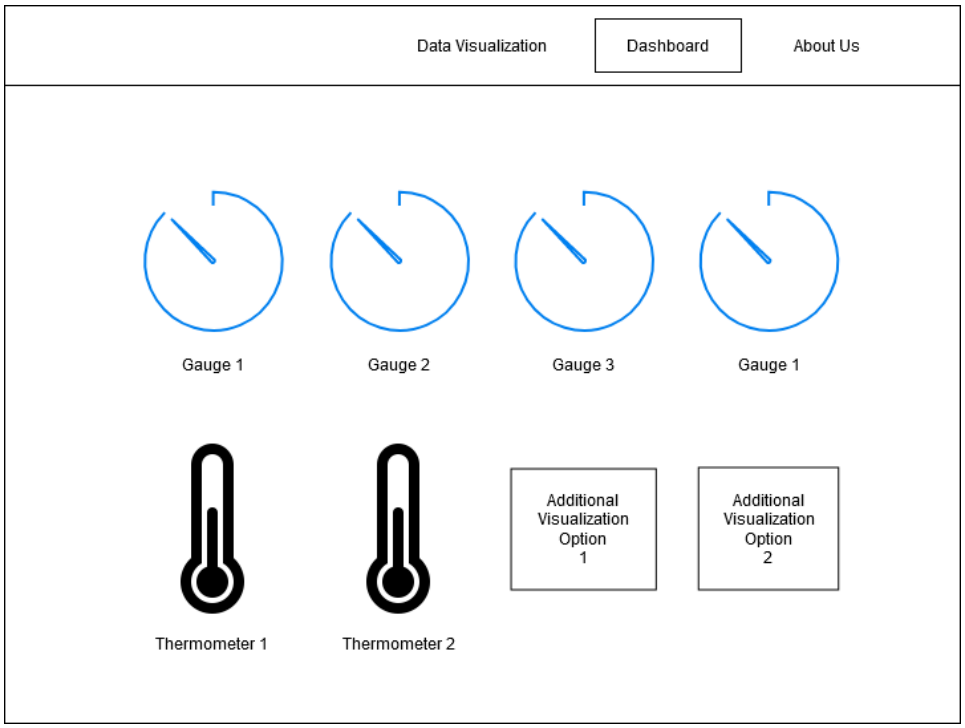


Figure 3.8: Prototype of a Dashboard Visualization Page.

acteristics is represented in Figure 3.8.

Analyzing Figure 3.8, we can see multiple gauges, this is due to the fact that gauges provide a simple and concise display of current data values, which is the main goal of a dashboard page.

On top of that, there are 2 different thermometers represented in the prototype. These thermometers are used in dashboards for the same reason that gauges are, since they are fundamentally identical to them. These are, however, generally applied to temperature observation, despite being applicable to any value that could be shown by a gauge. That being said, thermometers are still a very robust visualization option for current data displays, even though they are normally used only in specific cases.

Lastly, there are 2 blank visualization options represented in the prototype, which are more case specific. For example, if an user decides to incorporate a visualization option that is capable of going through the en-

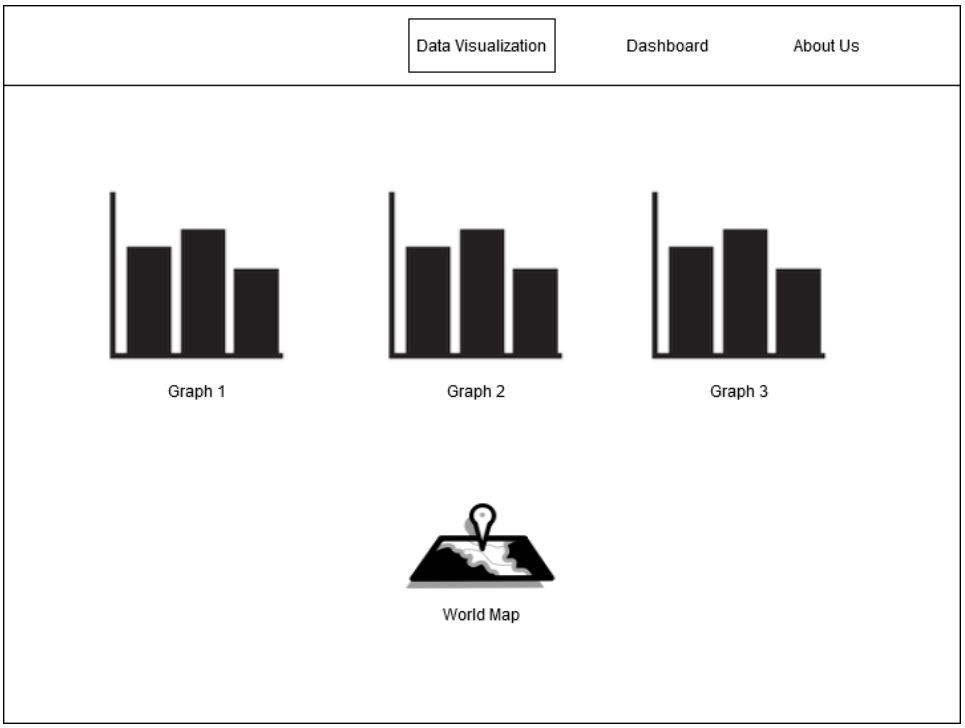


Figure 3.9: Prototype of a Stored Data Visualization Page.

tirety of a characteristic's data progression, a graph can be used to do so, if an user wants to display geographical positioning data, a world map can be displayed and if an user simply wants a more cut-and-dry current data visualization option, a text box could be added to any of those spaces.

Stored Data Visualization Page

In the case of a stored data visualization page, the main focus is to observe the entirety of data progression and facilitate decision making based on that data as much as possible. In order to do this, simple yet informative visualization options should be used, so that data can be observed quickly without sacrificing any of its information.

In Figure 3.9, a prototype for the stored data visualization page is represented.

Analyzing Figure 3.9, the main visualization tools used are graphs. This

is due to the fact that graphs are the best way of demonstrating data progression over time. There are 3 represented graph options in the prototype, but, clearly, the number of graphs should vary depending on the amount of characteristics to be observed, given their relevance in this context.

There is, however, a world map also represented in the prototype. This world map can be used in the case that the data progression is not only tied to time, but also space. What this means is that it can also be necessary to pinpoint geographical locations at the time characteristics were taken, in order to reach the right conclusions and solve the right problems.

Chapter 4

Project

In this chapter, the development of the solution that is the object of this thesis will be described. The solution aims to simplify and automatize the storage of the outputs of any sensor or group of sensors in databases, and its subsequent visualization through an online web tool.

4.1 System Architecture

As described previously, the purpose of the desired solution is to take whichever output a sensor or group of sensors provides, store it and display a simplified version of the obtained information through a web tool, without sacrificing any conclusions possible based on the stored data in the visualization process.

With this in mind, the overall system architecture can be seen in Figure 4.1.

By analyzing the provided system architecture we can see that the first step of the overall process is the acquisition of sensor data itself.

The sensor data may be acquired through one of two ways:

- Direct reception of the acquired data through socket communication;

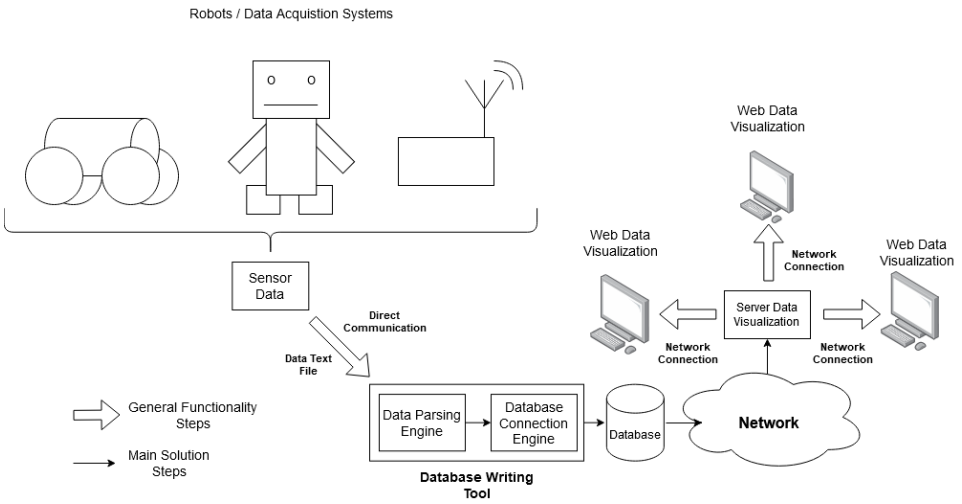


Figure 4.1: System Architecture.

- Data log containing all the information stored by any sensor over a period of time.

Once the data is acquired, it is then utilized by the "Database Writing Tool", which is the first part of the overall conceptual solution and is referred to as "Database Tool" from here on out.

This tool takes the output of a sensor or group of sensors, be it through log-files or data received through direct communication via sockets, parses it within the specifications set by an user, and then writes it to a table inside a database, based upon those specifications.

Once that process has been concluded, the acquired sensor data will be stored inside a database, which can be ran within a network, allowing other machines inside that network to access the stored data.

At this point, the second part of an overall solution comes into play. The "Server Data Visualization" portion of the overall system architecture correlates to the "Online Data Viewer", which is a web tool with the conceptual intent of facilitating the visualization of data stored inside databases.

4.2 Database Tool

The "Database Tool" was developed exclusively using the *Python* coding language and is responsible for reading data acquired from both log-files and socket communication, so that its information can then be written into a table inside a database of its user's choosing.

The "Database Tool" was conceived with the intent of automatizing the process of data parsing in a generally applicable way.

By creating the "Database Tool" with general applicability in mind, the need for individual scripts for sensor data storage disappears, which facilitates the workflow regarding sensors greatly.

The requirements necessary for installation of the "Database Tool" are detailed in Appendix A, along with the necessary steps for installation in both *Windows* and *Linux* operating systems.

4.2.1 Overall Functionality Summary

The developed "Database Tool" begins by acquiring the basic information of database to use and whether the data is to be acquired in real-time or through the use of log-files, by requesting inputs from the user, beginning the process of database table setup.

From there on and in the case of log-file data acquisition, the location of the log-file to be used is also requested from the user, along with the name of the table to write the data into. The user will then be asked to specify some of the characteristics of the data that is about to be interpreted, such as its header and its column data types, beginning the last process prior to the actual data parsing and writing to the specified table after that, which is setting up the date/time configuration of that data.

The process of date/time configuration is optional and is used to determine the time frame in which the data was taken. This can be done

either through conversion of time stamps[2] in the midst of the parsed data or through manual input of the date and time frame the data was taken on.

Once these initial processes based on user input have been concluded, the log-file data parsing can truly begin, finally parsing and writing the provided data into the specified table inside a database, based upon the parameters set by the user.

When it comes to the real-time data acquisition scenario, the processes that take place are largely the same, however no log-file location is requested from the user and, instead, during the process of data parsing a server socket will be opened, initiating the process of reading and writing the data into a table based upon the provided parameters as soon as a socket client sends data to it.

The flowcharts that summarize the overall functionalities of the developed "Database tool" are represented in Appendix B, along with other flowcharts that represent some of the processes that are part of those functionalities.

The following sub sections will further detail each of the summarized functionality steps of the overall solution, starting with the fundamental module necessary for it to work.

4.2.2 MySQL Connector

Besides the *Python* interpreter itself and its associated modules, the solution requires the "MySQL Connector" module to work properly. This is the module that allows the solution to connect to databases, even when they are based on "MariaDB".

The module is called at the beginning of the solution itself, and is used immediately to establish connection to any configured database, then utilizes that connection to start writing tables with the desired information.

[2] A time stamp, in the format of "Unix epoch time", is a record of time defined by the number of seconds that have passed since 1 of January 1970 at $00:00:00$.

```
#------ MariaDB Connection

mydb = mysql.connector.connect(
    host="localhost",
    user="root",
    passwd="maria123",
    database="ctd_data"
)
```

Figure 4.2: "MariaDB" Connection Configuration Code.

4.2.3 Database Table Setup

Initial Setup

Before running the solution for the first time, some initial configurations are required. Since the user, password and the host address relating to any "MySQL"/"MariaDB" session need to be specified in tandem before any connection is even attempted, it is necessary to add the mentioned characteristics in the beginning of the code itself. The snippet of code that should be altered to do this is represented in Figure 4.2.

Analyzing the code snippet, the user should be specified in the "user" section, password in the "passwd" section and host address in the "host" section, for any "MySQL"/"MariaDB" session. As mentioned previously this code snippet is located right after the dependencies are established. It is important to note that the "database" section does not need to be altered, since later on the users will be asked to specify which database to connect to regardless.

With the initial setup finalized, the first step towards writing tables in a database is the connection to the database itself, which is selected through user input of its name.

Basic Parameter Configuration

Once the initial setup has been concluded, the user is then prompted with a series of other inputs, the first of which defines whether the solution will

```
Please input the location of the desired log file. Type 'default' for a hard-coded test location.
C:\Users\Raymu\Desktop\Work\TEDI\Script e Dados Eng\ctd_process\ctd_data\20200316\CTD_20200316_00.txt
File location valid!

Log file obtained from C:\Users\Raymu\Desktop\Work\TEDI\Script e Dados Eng\ctd_process\ctd_data\20200316\CTD_20200316_00.txt
```

Figure 4.3: Successful File Location Console Output.

utilize pre-existent logs or socket communication to acquire its values. Besides the source of the data itself, not much varies between the consequential processes between these two options, but there are some differences of note that will be mentioned.

In the case of the socket communication method, the next step of the solution begins immediately, however if the data has been stored inside a file, the file's location is requested from the user. After the location has been provided through input, the solutions checks if the location is valid or not. If it is, the solution moves on to the next step, if it is not, the solution resets, prompting the user to provide a valid file location. An example of this step is represented in the Figure 4.3, which is the console output relating to a valid file location input.

Once the source of data has been established, it's then requested that the table name, in which the obtained data will be written to, be specified through user input. In order to make sure that the table name provided does or does not belong to an existing table in the database, the solution utilizes a "MySQL" query to acquire the names of every single table that already exists inside the database selected previously. Whether or not the table name provided correlates to an existing table's name changes the next step of the solution's processes.

Once its been established the name of the table to be written to and whether or not a table of that name already exists, the solution can work in one of two ways.

In the case that the table name already exists in the selected database, the user is then prompted on whether the data that's about to be parsed

should be added to the existing table or if the existing table should be cleared in order to accommodate the new data. It is worthy of note that, if the user decides to add the data to an existing table, the process of data parsing will begin almost immediately and the number of columns inside that table must be in accordance with the amount of columns the parsed data presents, otherwise the solution will not work.

On the other hand, if the table name does not exist in the database, this step of the solution is skipped entirely, creating a new table in the process, before advancing towards the next step.

This was done so that users can have the choice of adding additional data to a table that already exists without altering any of the previous data, making it possible to create a continuous flow of data acquired from different log-files or direct data reception via sockets if needs be.

When it comes to refreshing the selected table, this functionality was added with efficiency in mind, given that it streamlines the process of deleting and creating new tables immediately, if some wrongful parameter was set for the previously existent table with the same name.

Table Header Acquisition

After the basic parameter configuration has been finished and if the user decides to add to an existing table, that table's header is acquired through a "MySQL" query which is then utilized to begin the data parsing process, completing the process of table header and column data type configuration.

On the other hand, if the table the user is working on was just created or refreshed, that user is then asked to specify if there is an header in the log-file/socket data that is to be interpreted. If the header does not exist in the midst of the data, the user is then asked to specify each individual table column name. Each table column name input is confirmed by pressing the "Enter" key and inputting "\0" finishes this step of the operation.

In the case that an header does exist in the desired data, the last described step is skipped and instead a different prompt will be given to the user requesting the exact name of a characteristic present in the header of the log-file/socket data.

Column Data Type Definition

Once the header has been established or a characteristic's name present in the header has been specified, for a new or refreshed table, it is then requested that the user specifies which table columns should have their values considered "characters" and which should be considered "decimals", based upon the data that is to be acquired. In order to do so, 2 prompts are presented to the user, the first one requesting the names of all the headers whose columns should be considered "characters" and, once that process is finished, which are the headers whose columns will be defined as "decimals". Each of these processes are considered complete whenever an user inputs "\0", in the same fashion as the individual table column name acquisition process. An example of these prompts, as well as different possibilities for user inputs, is represented in Figure 4.4.

It is important to note that all the headers that are not listed by the user during these prompts, will have their respective column's values defined as "floats".

The reason why this entire process is done stems from the fact that values such as those seen in time stamps and geographical coordinates are too large to be able to be defined as simple "floats" when it comes to the creation of tables inside databases. When defined as "floats", the aforementioned time stamps and geographical coordinates have their decimal values truncated when written to a table and, due to the precise nature of how the tables are dynamically filled through "MySQL" queries and the goal of the project itself, such is not acceptable. The same principle applies to characters, since

```
Please indicate which columns should be considered as characters (Column names). Write '\0' to finish the process.Names
Please indicate which columns should be considered as characters (Column names). Write '\0' to finish the process.Time&Memory
Please indicate which columns should be considered as characters (Column names). Write '\0' to finish the process.\0
Columns whose values will be considered chars: ['Names', 'Time&Memory']
Please indicate which columns should be considered as extra-long decimals (Column names). Write '\0' to finish the process.Timestamp
Please indicate which columns should be considered as extra-long decimals (Column names). Write '\0' to finish the process.\0
Columns whose values will be considered decimals: ['Timestamp']

Remaining columns will be defined as floats.
```

Figure 4.4: Column Data Type Configuration.

the "float" data type does not properly compute said characters. Once this process is finished, the table header and column data types for new/refreshed tables configuration is complete.

On the other hand, conceptually, this sequencing of events is done due to the fact that "decimal" and "character" values output by sensors are more often than not the exception rather than the rule, given that these various are normally numerical with not too many decimals places, which the "float" data type is compatible with. By defining all the columns that weren't mentioned as "floats", it saves the user the time of having to go through most of the header names one by one.

The algorithms that include the mechanisms that were described in this sub chapter will be introduced later, when the acquisition of data itself is described.

4.2.4 Date/Time Configuration

Initial Processes

Once the overall "database table setup" process is complete, for both new or altered tables, the user is then prompted on whether or not the date/time configuration of the acquired data is necessary.

This part of the solution begins by creating (if it didn't exist before) a table by the name of "table_data" inside the database, which includes columns for table name, date, start time and end time of the obtained data. This table is used to store the basic configuration of date and time for the interpreted data relating to a specific table, without changing the table itself.

Once the “table_data” table has been setup, the name of the table that is being configured is immediately added to the first column of the “table_data” table , regardless of whether or not there is going to be any date and time information to add.

If the user decides to not add/alter the date and time configuration for a certain table, the process of log-file/socket communication data parsing begins immediately.

Real-Time Data Acquisition Case

In the case that the user does decide to add/alter the date and time configuration for a certain table and in the case of socket data reception, if the user specifies that the data to be acquired does not include any values relating to time stamps, the current date and time are then obtained automatically from the operating system’s date and clock, being then added added to the “table_data” columns relating to date and start time of the obtained data respectively.

This process is then followed by the parsing and writing of received data into the preferred table inside a database. During the data acquisition and parsing, the column relating to the end time of the received data inside “table_data” is updated for every new line received through socket communication towards the table the data is being stored into, using the operating system’s current time to do so.

This is done so that if either the client or the server communication is interrupted, the time at which the data stopped being received remains correct, regardless of when it happens.

On the other hand, if the data does have time stamp information, the solution will utilize that instead of the system’s clock, through time stamp conversion, whilst also adding the converted time to the rows of the selected table that is being written into. Firstly, two new columns are added to the

table in which the received data is going to be written into, namely the "Date" and "Time" columns.

Once that has been finished, the user is asked the name of the column that includes the time stamp, which is utilized when this component of the solution begins working in tandem with the parsing of the received socket data. During this process and for every new line of data, the specified time stamp column value is converted into date and time, adding these values to the "Date" and "Time" columns respectively.

This date and time is not only added to the table that receives the data, but also to the "table_data" table in which the initial converted date and time are added to the "Date" and "StartTime" columns relating to that table, whilst subsequent time conversion from the following lines added to the "EndTime" column of that same table.

The usage of the systems' clock instead of time stamp conversion, when there is none, is due to the fact that the real-time data acquisition implementation of this process is, as the name implies, designed to receive data as soon as it is ready to be output.

Although not perfect, due to delays between both the time a characteristic is taken by a sensor and the time it is output by that sensor, adding to that the time between the data being sent by the client and received by the server, the systems' clock is still capable of providing a fair idea of the time frame data was taken when no other data is provided in this scenario.

It is worthy of note that all the described date/time conversions being made to the time stamps are done in regards to the Greenwich Mean Time (GMT) time zone.

Log-file Data Acquisition Case

In the case of log-file data acquisition, after confirming that there should be a configuration for the date/time of the received data from a log-file

to a table, the user is then prompted to define whether or not the log-file itself contains data relating to the date/time configuration, in the form of time stamps. Like in the described socket communication implementation, if time stamps do exist inside the log-file, the "Date" and "Time" columns are added to the table in which the data is going to be written into and the user is then asked which of the columns contains said time stamps, so that the process of log-file parsing may begin.

During the process of log-file parsing, the time stamp inside the first interpreted row of values in the log-file is converted into time and date and is utilized to fill the "table_data" columns relating to start time and date respectively, whilst also adding that data to the generated "Date" and "Time" columns of the table being written into in the same way.

During the process of log-file interpretation and data storage and for every new time stamp beyond the first one, the conversion of the date and time are used in an identical way, being that the converted time is now added to the "table_data" column relating to the end time instead of the start time column. This process was implemented for similar reasons to those in the socket communication data acquisition case, given that an user could terminate the process of reading and storing log-file data at any time. On that note and when it comes to the "Date" and "Time" columns beyond the first line of data, the process is still the same for every row after the first one.

On the other hand, if there is no actual date/time configuration inside the log-file and the configuration of date/time of the interpreted data is necessary, the user is,in that case, asked to manually specify the date, start time and end time of the data inside the log-file that is about to be interpreted following the end of this process ("Year-Month-Day" and "Hour:Minute:Second" format for date and time, respectively).

An example of the previously described prompts and user inputs are

```
Does the log-file have date and time information?
(0 - No; 1 - Yes)
1
Is the date and time information in the log-file? (timestamp format)
(0 - No; 1 - Yes)
0
Please specify the date in which the data was gathered. (YYYY-MM-DD format)
2020-05-08
At what time of day did the data begin being acquired in the log-file? (HH:MM:SS format)
12:00:00
What time did it end? (HH:MM:SS format)
13:11:32
Additional Configured Table Data:

Table Name: test

Date of Data: 2020-05-08

Data acquision started at 12:00:00 and ended at 13:11:32.
```

Figure 4.5: Date/Time Configuration Log-File Prompt and User Input Example.

represented in Figure 4.5.

The reasoning behind this entire process, for both log-file and real-time data acquisition, is to setup temporal data that exists inside or outside the acquired data itself, either through time stamp conversion or manual input, into the table and database that data is being stored into. By providing this option, if the final stored data includes characteristics referring to date and time frames and/or includes an additional table with only the date/time configuration data, the overall data could be visualized in different ways based on temporal specifications that would not be possible without the described configuration.

4.2.5 Log-File/Socket Data Parsing

In this sub section, the main part of the developed solution, the log-file/socket data parsing, will be described. This includes not only the functionalities behind both the interpretation and writing of the received data, but also the requirements that the sent data must abide by. These requirements

have been made has generally applicable as possible, but some concessions were necessary in order to achieve the desired functionalities.

Log-File Structure

When it comes to the structure of log-files usable by the "Database Tool", the data itself should be stored inside a text file first and foremost, so that the *Python*-based solution can open and read its contents. When it comes to how the file data itself is presented, however, there is a large degree of freedom. The text file should contain columns (with or without an header row) separated by either a blank space, commas or a combination of both.

This was setup in this way to provide parsing compatibility to the largest amount of different types of log-files, within reason, by the tool.

If there is in fact no header anywhere inside the text file, as mentioned previously in this section, the user can name each of the columns inside the log-file so that they can then be written into a table.

It's important to note that, in the case of header rows in log-files, where the header row is located inside the text file does not compromise the solution as long as it is indeed there.

This is due to the fact that the solution will do a quick run-through of the file in search of an header row, if such does exist inside the file, and utilizes the first one it sees to define the header of the table it will write into, before doing a second run-through to acquire all the values inside the file.

This run-through is necessary given that, otherwise, the lines of data that existed prior to the first header line available would be ignored, since the table in which the data would be written into would not have the necessary header data to be defined yet and, as such, no actual data could be written into it at that point.

Lastly, if there are certain lines that end up containing only a single

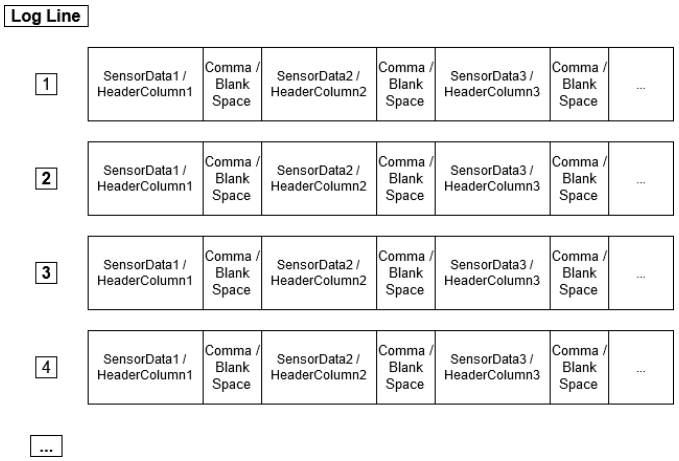


Figure 4.6: Possible Log-File Layout (Line by Line).

column or are empty when perceived by the solution, they will be deemed as "clutter" and ignored. There are some other data output scenarios that should result in lines that are to be considered "clutter" as well, however these are very dependent on the system that writes the logs to text and this solution aims at being as generally applicable as possible. Extra conditions can easily be added to the solutions code for specific scenarios in this context, but ideally the log-file should be filtered to include as little "clutter" as possible.

The optimal way of structuring a log-file on a line by line basis is represented in Figure 4.6.

Socket Communication Requirements and Message Structure

If data is being sent in real-time through a socket, the main requirement is that the data is sent in the form of an array sequentially, where each of the values that comprise it relate to a value inside a row's column. This array, however, needs to be converted into a string so it can be sent from a socket client to the socket server included in the solution.

It is acceptable that an user chooses to send an header among the data rows, however it isn't recommended since, due to the nature of how the real-

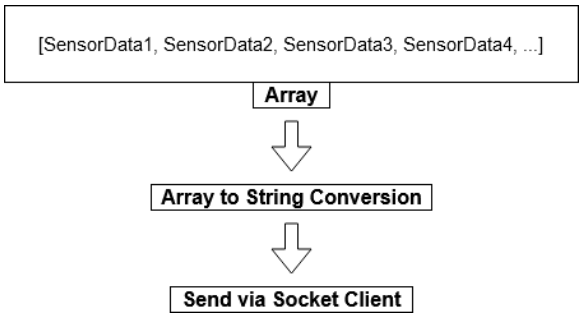


Figure 4.7: Optimal Individual Socket Message Layout.

time implementation of the solution works, the data will only start being written to a table once an header is interpreted.

The optimal way of structuring sensor data to send to the socket server from a socket client is represented in Figure 4.7

Initial Process Description

When it comes to the actual Log-file data parsing, the process begins in the previously described "Database table setup", since there are few conditions that need to be met during the process of log-file/socket data parsing.

In order to begin the process of data interpretation from any source, it's first necessary to know whether or not an header has already been established for the data that is about to be received from whichever source. If an header was acquired via manual input or "MySQL" query, the second characteristic from the header is stored as a variable, being then used to check if certain lines of received data contain header data that is unnecessary and should be discarded.

If the header is inside the received data rather than acquired through the aforementioned options, the variable relating to the observed characteristic will be acquired from user input and used as a way of finding the line relating to the header inside a file. The discovered header will then be used towards creating a new table, as well as finding and eliminating possible replicas of

that same header line along the log-file.

Data Acquisition

Once the data acquisition processes do begin and in the case of the source of data being from a log-file, the source file is opened a first time in order to find an header line if one hasn't been acquired yet, step which is skipped if there already is a defined header for the table that going to be written into a database.

Regardless of whether or not the last described step was skipped, the file that contains the desired data is opened (be it for the first or second time) and the lines of data inside it start being read sequentially, only stopping either through user input or if the end of file was reached. The lines are read one at a time, as mentioned previously, and the values inside them are immediately inserted into a buffer based upon comma and/or blank space separation in between these values.

Once that's done, a verification is done in order to assess if the buffer has a length superior to that of a single value, in order to make sure the saved buffer doesn't correlate to a line with no actual data.

If it is indeed a buffer with actual data, it's then verified if the buffer contains any data that could be that of an header, by checking if any of its values is equal to the variable pertaining to the observed characteristic that was either acquired from user input or automatically. On the other hand, if it is not a buffer containing header names, the value line parsing process begins.

In the case of the real-time data reception through socket communication, the process is very similar to that of the log-file data acquisition, with some key differences. Since the data is acquired through server request and client response, instead of reading lines and given that the data is already in the form of a string that can be converted into an array, it isn't necessary

to separate the values from each other since they can all be easily inserted into a table if necessary.

The other key difference is that, since there is no file that helps us predict how much information will be received and due to how unpredictable the received data from each request can be, if the user tells the solution that there is a header in the midst of the received data in real-time, the solution will wait until the actual header line is received before writing any values to a table in a database.

It's also important to note that, even in this context, the mechanism relating to the observed characteristic's variable is used, which means that all the header lines received beyond the first will be ignored.

Header/Value Line Parsing

When it comes to the parsing of header/value lines, the process is the same for both the real-time and log-file data options.

The process begins by identifying if the line is supposed to be treated as an header or not, based on the described processes that precede it. If it is indeed an header, it will go through all the items inside a line, being that the first one will used to create the table itself through a "MySQL" query, whilst the other ones are added sequentially, also utilizing "MySQL" queries, to the created tables' header.

Using the data provided by the data type configuration done in the "database table setup", the process determines how each header should be configured when inserted into a table so it can then correctly sort the received values in its columns. The columns can be defined as "decimal", "float" or "character", based upon the provided header names' data type configuration.

If the line provided isn't an header line, then the process acts on it as a value data line. The values are added in the same order that the header

names were added, so that these values can be correctly associated with their actual headers. The first value in a line is used to define a new row inside a database table through "MySQL" queries, whilst the following ones update that same row with new columns with their own values, also through "MySQL" queries. The inserted values have their data type defined inside a table by the header of the column they were inserted into.

Overall Log-File/Socket Data Parsing Algorithms

The algorithm that represents the process of log-file data parsing is represented in Algorithm 1, which also includes the functionality of the "Database table setup" mechanisms. In the same vein, the algorithm that describes the process of real-time data parsing is represented in Algorithm 2.

4.3 Online Data Viewer

The "Online Data Viewer" is a web-based data visualization solution, developed using HTML, PHP and *JavaScript* languages, alongside CSS to define the fixed visual characteristics of the overall web site. The solution is used in order to parse table data received from databases and facilitate its interpretation through robust data filtering and customization options, along with varied visualization possibilities for both old and current data.

This part of the solution can be ran in the same server in which the databases are stored and can then be accessed by any computer running inside the network. It is important to note that the "Online Data Viewer" uses data stored inside tables in database, tables which do not need to necessarily be written by the "Database Tool".

The "Online Data Viewer" itself provides a customizable dashboard that focuses on showing the current status of all the variables that are stored in a table inside a database, database and table which are chosen by the user in

Algorithm 1: Database Tool Log-File Parsing Algorithm.

```
   Result: Data written to selected table in database
 1 Header_Check ← 0;
 2 if New table then
 3 |   if Header in received data then
 4 |   |   input ObservedCharacteristic;
 5 |   |   call Column Data Type Configuration;
 6 |   else
 7 |   |   Header_Line[] ← call Manual Header Acquisition;
 8 |   |   call Column Data Type Configuration;
 9 |   |   ObservedCharacteristics ← Header_Line[1];
10 |   end
11 |   if Alter date/time configuration then
12 |   |   call Date Time Configuration;
13 |   end
14 else
15 |   if Alter date/time configuration then
16 |   |   call Date Time Configuration;
17 |   end
18 |   Header_Line[] ← MySQL Query Fetch table header;
19 |   ObservedCharacteristics ← Header_Line[1];
20 |   Header_Check ← 1;
21 end
22 Data[] ← Open File Location;
23 Current_Line ← 1;
24 while not All lines read do
25 |   Read Data[Current_Line];
26 |   Data[Current_Line] ← Data[Current_Line].ToBuffer();
27 |   if Header_Check == 0 && ObervedCharacteristic in
   |    Data[Current_Line] then
28 |   |   Call Header Line Parsing;
29 |   |   Header_Check ← 1;
30 |   else
31 |   |   if Header_Check == 1 && len(Data[Current_Line]) > 1
   |   |    then
32 |   |   |   if not ObservedCharacteristic in Data[Current_Line]
   |   |   |    then
33 |   |   |   |   Call Value Line Parsing;
34 |   |   |   end
35 |   |   end
36 |   end
37 |   Current_Line++;
38 end
```

Algorithm 2: Database Tool Real-Time Data Parsing Algorithm.

```
   Result: Data written to selected table in database
39 Header_Check ← 0;
40 if New table then
41 |  if Header in received data then
42 |  |  input ObservedCharacteristic;
43 |  |  call Column Data Type Configuration;
44 |  else
45 |  |  Header_Line[] ← call Manual Header Acquisition;
46 |  |  call Column Data Type Configuration;
47 |  |  ObservedCharacteristics ← Header_Line[1];
48 |  end
49 |  call Date Time Configuration;
50 else
51 |  if Alter date/time configuration then
52 |  |  call Date Time Configuration;
53 |  end
54 |  Header_Line[] ← MySQL Query Fetch table header;
55 |  ObservedCharacteristics ← Header_Line[1];
56 |  Header_Check ← 1;
57 end
58 Open Socket Communication;
59 while Communication open do
60 |  Current_Line ← Server-Side Data Request;
61 |  Read Current_Line;
62 |  if Header_Check == 1 then
63 |  |  if ObservedCharacteristic in Current_Line then
64 |  |  |  if len(Current_Line) > 1 then
65 |  |  |  |  call Header Line Parsing;
66 |  |  |  end
67 |  |  end
68 |  else
69 |  |  if ObservedCharacteristic in Current_Line then
70 |  |  |  call Header Line Parsing; Header_Check ← 1;
71 |  |  end
72 |  end
73 end
```

the web tool itself. It also includes additional pages, conceived to be more autonomous system-specific, that allow for data visualization centered more around data progression rather than latest values.

On that note, the "Online Data Viewer" was conceptualized with ease of access in mind, given that, as previously stated, only one computer/server needs to be running the solution for it to be accessible by anyone who needs it, whilst creating a concise and informative data display option through the use of customizable dashboards and pages orientated around data progression observation.

4.3.1 Solution Development as a Server

Before development began, the machine used had the necessary RDBMS installed ("MariaDB") in conjunction with a web server, namely the "Apache" server.

In order to utilize the "Online Data Viewer" when a server has it running on the same network, all that needs to be done is to connect to the server using its Internet Protocol (IP) version 4 address as the URL.

On the other hand, if an user wants to run the solution as the server itself, a web server alongside a Hypertext Transfer Protocol (HTTP) daemon needs to be installed in a machine.

A HTTP daemon is executed in the background of a web server and is responsible for carrying out the process of waiting for and decoding any HTTP request, acquiring any information necessary for said request, such as data stored inside a specific database, for example [62].

When it came to developing the "Online Data Viewer", the "Apache" HTTP server was utilized, which includes both a web server and a HTTP daemon [63].

It is important to note that, without a web server such as the "Apache" HTTP server running in the background, the PHP development language

cannot be interpreted. This coding language is essential for most of the features that will be described during the development of the "Online Data Viewer".

A full run down on how to install and run the "Online Data Viewer" in a server is provided in Appendix A.

4.3.2 Bootstrap

Before detailing the development of the "Online Data Viewer", it is important to mention the framework used to develop the solution, namely "Bootstrap".

The "Bootstrap" framework was utilized for development given that it enables the creation of responsive, mobile friendly web sites, organizing all visual components in a grid system that adapts to the platform used to access them.

As such, regardless of which platform is used to access the "Online Data Viewer", it is to be expected that the web site will run smoothly, and all the options inside it will be usable by anyone [64].

There are also some visualization options provided by "Material Design for Bootstrap" (also known as "mdBootstrap") [65], which will be used and shown later in the solution's development description.

4.3.3 Dashboard & Settings

In order to facilitate the visualization of data obtained from a database in real-time, a "Dashboard" page was created with the intent of observing the latest values being stored in specified tables inside any database.

Since the nature of these values differs greatly depending on the characteristics they represent, be it between tables or table columns, it is important that the user can customize how the data should be visualized given the context it is in. As such, a "Settings" page was created as a companion to the

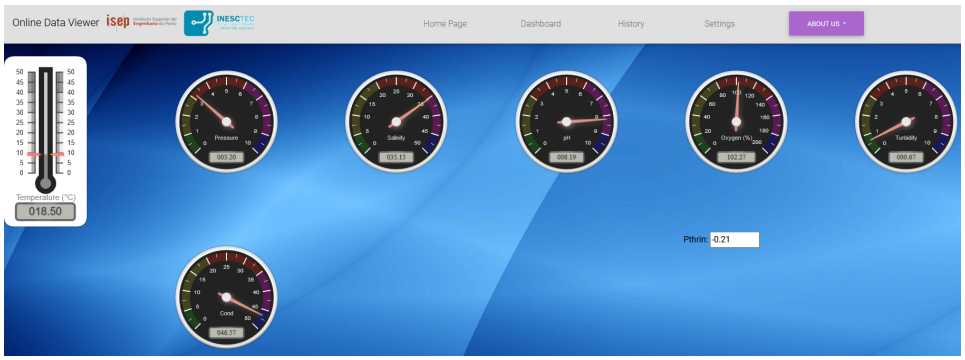


Figure 4.8: Dashboard Concept (Page Start).

dashboard itself, and allows users to customize how and what visual tools are displayed inside the dashboard.

By allowing users to determine what they want to see in the "Dashboard" page, most visual clutter that could exist in that page can be minimized in accordance to their perspective, providing a form of data display that is both informative and dynamic.

Dashboard

When it comes to the dashboard and since the design of a visualization tool is one of its most important features, an early concept build of the dashboard page was created. This conceptual dashboard utilized data from a fixed table in a database and the visualization tools could not be customized. This was done so, when the real dashboard were to be developed, the main design concept was established and workflow on it would be facilitated.

In Figure 4.8 the start of the conceptual dashboard is represented and in Figure 4.9 the end of the conceptual dashboard, which includes the world map integration, is shown.

As for the finalized dashboard itself, there are a few different visualization options that an user can utilize for data display.

Whether or not each of the following visualization options is displayed, along with the data that is displayed, is completely customizable by the

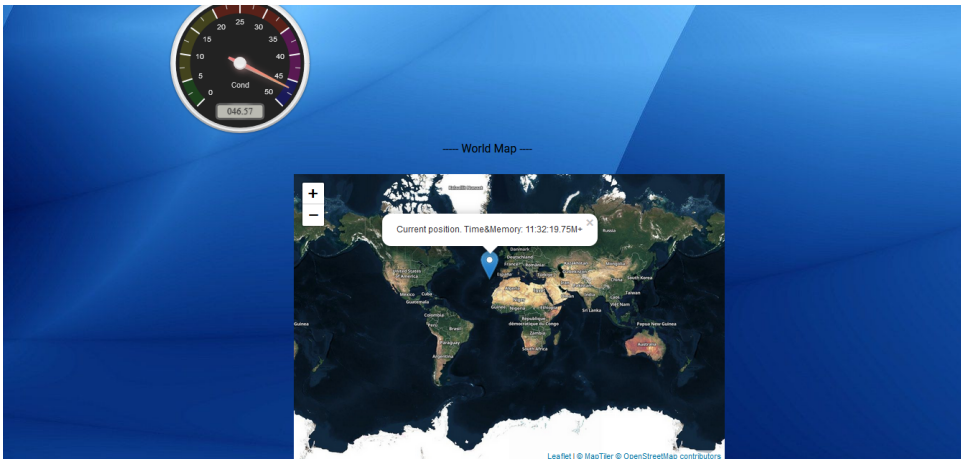


Figure 4.9: Dashboard Concept (World Map).

user through the "Settings" page, which will be described later on in this sub section.

Firstly, there are 6 gauges available for use, 1 of them linear and the other 5 radial. The linear gauge is meant to be used to represent temperature, given that it takes the form of a thermometer. As for the radial gauges, they resemble velocimeters, which entails a fair degree of flexibility on which variables can be represented by them. That being said, variables with negative values aren't best suited for radial gauges, unlike linear ones.

The design of the gauges utilized were achieved through "Canvas Gauges" templates, which are free to use [66].

It's important to note that the range of shown values varies depending on the data acquired from the database to be displayed. To do this, during the data acquisition process (based upon the provided configuration) all the values are parsed, up until the one that is going to be shown, and the highest value out of all of them is saved. This highest value is then used to assess which option of range should be used in visualization.

The described process is applied to radial gauges, where only positive values are normally expected. When it comes to linear gauges, for positive values the process is the same, however, if the value to be represented is

negative, the visualization uses a single range for those values. The possible ranges are represented in Table 4.1, being that the validity checks as the code runs are done in the same way has the ranges are ordered in said table from top to bottom.

Value Case	**Displayed Range**
Last Value < 0*	[-150,-135,-120,-105,-90,-75,-60,-45,-30,-15,0]*
Highest Value $\leqslant 10$	[0,1,2,3,4,5,6,7,8,9,10]
Highest Value $\leqslant 50$	[0,5,10,15,20,25,30,35,40,45,50]
Highest Value $\leqslant 100$	[0,10,20,30,40,50,60,70,80,90,100]
Highest Value $\leqslant 200$	[0,20,40,60,80,100,120,140,160,180,200]
Highest Value $\leqslant 1000$	[0,100,200,300,400,500,600,700,800,900,1000]

Table 4.1: Possible Gauge Display Ranges. *-Linear Gauge Only.

Secondly, the "Dashboard" page includes 4 text boxes that can be filled with whichever characteristic that an user would prefer to configure there, being that these text boxes can only be read once configured. The only limitation that the text boxes have is that they can only fit up to 11 characters, any amount of characters over that would still be represented, but may not be fully legible.

These text boxes were defined through embedded *JavaScript* features.

Thirdly, 5 graph displays can be used inside the "Dashboard" page. Out of those graphs, 4 of those are used for individual variable progression observation. These graphs allow for "X" and "Y" axis variable customization, along with the amount of latest entries to be used from the fetched data.

As mentioned previously and as will be detailed later, this is all done through the "Settings" page, where it is also possible to customize how many of these graphs can be represented per row, from 1 all the way to 4.

The fifth graph is used to compare the progression of multiple variables over time. This graph also allows for "X" and "Y" axis variable customization, but can show up to 3 different variables in the "Y" axis. This graph, given the amount of data that it can contain, is always represented in its

own row, regardless of the previously mentioned “graph per row” setting.

All of the graphs were designed using “mdBootstrap” [65].

Lastly, there is a world map display that shows the last recorded positional value from the data set acquired after the configuration has been established by the “Settings” page. Also, it is possible to configure a time stamp marker for the aforementioned positional value if the user chooses to.

The world map was implemented through the incorporation of “Leaflet”, which is a recurring feature of the overall solution and will be further described later in this section [67].

All the data visualization options shown for the “Dashboard” page were setup with immediate data representation in mind, so that users can get quick feedback regarding the current state of all the variables currently observed or, in the case of the graphs, the most recent data progression tendencies of specified variables.

Settings

In order to customize the data display shown in the dashboard, the “Settings” page was developed.

When the “Settings” page is opened, it is requested that the user specifies the host, username and password of the RDBMS that is about to be used for data visualization, so that the basic connection to it can be setup for data acquisition. Once these inputs have been submitted, they are saved within text files inside the server and the main portion of the “Settings” page is opened.

If the data relating to the basic connection to the RDBMS has already been set, the user can skip this step entirely and proceed to the main part of the “Settings” page instead, which is necessary feature to add, given that one of the main objectives behind the overall solution is to facilitate workflow as much as possible

When inside the main portion of the “Settings” page, the user is prompted to specify which of the databases inside the configured RDBMS should be used. The option selection is done through a drop down menu, whose values are acquired through a “MySQL” query.

Once the user chooses a database, the “Settings” page refreshes itself and, utilizing a “POST” type form submission into the page itself, does a second “MySQL” query utilizing the selected database to obtain all the names of the tables inside it. Through the use of these acquired table names, another drop down option selection menu is introduced to the user, in order to select the table which contains the desired stored data to be used in the dashboard.

Once a table has been selected another “POST” type form submission is made, but this time the chosen table name is used in a “MySQL” query that acquires all the names of the columns inside the selected table so that the user can then specify which visualization options are to be used for each characteristic inside the table.

This is done through radio buttons, whose values are defined by the fetched column names, for each of the available visualization options described previously inside the “Dashboard” page. On top of the radio button selections generated through the fetched column names for each visualization option, there is a “None” radio button selection that leads to that specific visualization option to not be displayed inside the final customized dashboard.

In the specific case of the first 4 graphs, both the “X” and “Y” variables to be represented can be defined by the user. If the “Y” variable is not defined, the graph will not be shown, however if the “X” variable is not defined but the “Y” variable is, the graph will be represented in the dashboard regardless, but the “X” axis of the graph will be filled with a received data entry counter.

The reasoning behind this is that, in the rare case that the characteristics being visualized do not have something similar to a time stamp in the midst of the overall stored data to correlate them with time, the variable data progression itself can still be observed for any chosen characteristic.

That being said, it's possible to specify the amount of table entries used for data representation through graphs in the dashboard, being that leaving this section blank is interpreted as if all the table entries should be used. For example, if the user defines "500" as the number of table entries for one graph, the last 500 table entries fetched will be the ones that are used by that specific graph in the dashboard and all the previous ones will be ignored.

When it comes to the number of these graphs that are displayed per row, a slider is shown to the user that allows for the selection of 1 to 4 graphs to be represented in every row.

Regarding the representation process behind this option, there are a few different possibilities to take into account:

- 1 Graph per Row

  If the user chooses to represent a single graph per row, the HTML representation will be done by opening and closing a row division every time a graph is to be represented, creating multiple rows for multiple graphs.

- 2 Graphs per Row

  If the user decides to represent 2 graphs per row in the "Dashboard" page, the solution will count how many graphs have values to be represented. If the graph to be represented is the first or third one, a new row division will be opened, whilst if it is the second or fourth one, the row division will be closed after the graph is added.

- 3 Graphs per Row

Similarly to the previous option, if the user selects to represent 3 graphs per row, the solution will count the amount of graphs that are being represented. If the graph is the first or fourth one to be shown, a new row division will be opened for the graph, if it is the second one, a new graph will simply be added to the row division and if it is the third one, the graph will be added to the row division followed by the closure of said division.

- 4 Graphs per Row

  In the case of the "4 graphs per row" option, a single row division is opened for all the 4 possible graphs that can be represented in the "Dashboard" page.

The fifth graph that can be represented is a comparison graph that is shown in its own row. This graph also includes "X" and "Y" variables that can be defined by the user, difference from the other graphs being that up to 3 different variables can be selected for the "Y" axis for comparison purposes. This graph largely follows the same representation rules as the previous ones, including the specification of table entries to be used, with the added restriction that at least 2 variables need to be set for the "Y" axis, otherwise it won't be represented given its purpose.

Up until this point, every single one of the visualization setup options can have their units of measurement specified. This option was added in order to improve the interpretation of acquired variables in the case that the header that specifies them does not hold that information. This features appends, in between parenthesis, the specified units of measurement for that visualization option to the name of the variable used for data representation in that same option, when shown in the "Dashboard" page.

The world map display option also has some singularities to take into account. Namely, if either the latitude or longitude of the desired position

for a marker is not specified, the world map will not be shown inside the dashboard. Besides that, there is an visualization option to add a time stamp to the marker itself, having an additional selection to fetch the current time of the server that is running this "Settings" page itself, on top of all the standard selections that exist for the other visualization options.

This feature was implemented in the case that the table from which the geographical data is being acquired from is being updated in real-time and does not have any sort of reference to time in any of its columns. By associating the current time of the operating system that runs the web page to the current geographical position being observed in this scenario, additional awareness in regards to time and space of the overall data can be provided to the dashboards' viewers that otherwise would not be possible.

Note that the world map display does not include measurement unit definition for its variables, given that its purpose is to simply situate the previous data values in both time and geographical space, rather than to measure any sort of characteristic.

An example of how the beginning of the "Settings" page looks like after the second form submission is represented in Figure 4.10. If any visualization option does not have any data associated with it after the submission has been made, said visualization option will not be displayed in the "Dashboard" page.

Once the user defines which characteristics (or lack there of) should be represented in each individual visualization option, these can then be submitted into the "Dashboard" page. This includes all the selected data during the described "Settings" page processes.

Upon submission of the selected visualization options, the dashboard is opened, including the desired visual configuration by the user. Since the submitted characteristics are essentially specifications of the database, table and column names from which to visualize data from, "MySQL" queries can

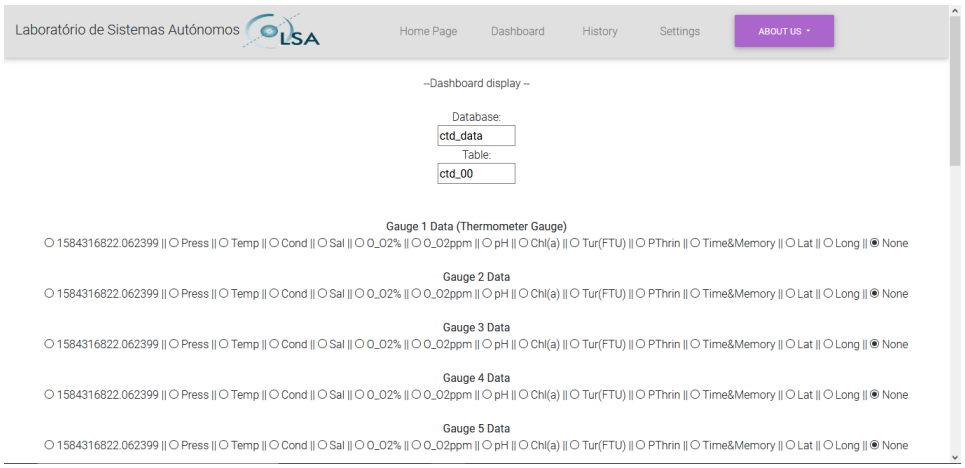


Figure 4.10: Example of the "Settings" page (Start of Page, After 2 Form Submissions).

be made depending on the desired visualization case and value acquisition.

The submitted characteristics, once received on the side of the "Dashboard" page, are stored in server-side text files and whenever it is opened (be it after closing the window or changing page) from there on out, the visualization options stored in said text files are the ones used. The stored data can be changed by submitting a new visual setting to the dashboard through the "Settings" page itself.

4.3.4 History

The aforementioned "Dashboard" and "Settings" pages were created with the intent of facilitating visualization of data acquired in real-time, given that the values shown are the latest ones stored in a database. That being said, it's also important to visualize data that had been fully stored previously in its entirety.

The development of a visualization option envisioned for the observation of data taken over time, rather than primarily the latest values of said data, resulted in the "History" page. This page is very different to that of the "Dashboard" page, in terms of customization. Where the "Dashboard"

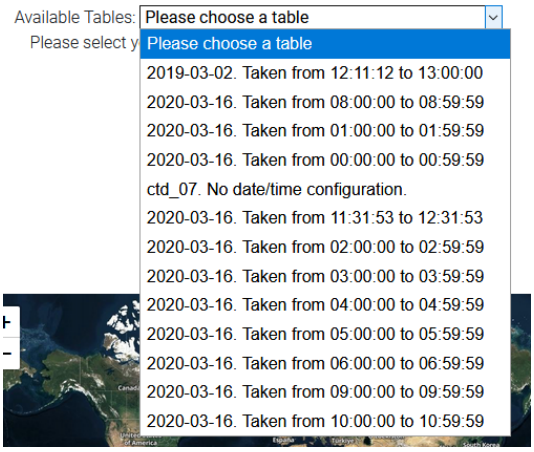


Figure 4.11: "History" Page Table Options Example.

could be customized visually, the "History" page has preset visualization layouts, with different presets for different autonomous systems, that display all of their data available for representation. Once the user chooses which of the preset autonomous system layouts should be the focus of visualization, the page relating to said layout will be opened.

An an example and once opened, one of these pages, which relates to the "TowFish" autonomous system, prompts the user to select which table's data should be accessed, through a drop down menu and within the database in which its data is stored. This drop down menu normally defines the different tables not by their actual name, but by the date and time period that they were taken from. This date and time period data is fetched directly from the "table_data" table mentioned during the "Database Tool" description in this thesis. If there is no date/time configuration for a certain table, then the page shows the actual table name instead, exposing to the user that there is no date/time configuration for said table. An example of these different options can be seen in Figure 4.11.

Given that this "History" page layout is made to be more case-specific than the "Dashboard" page, its configuration features were implemented in the context of the extra degree of solution freedom provided by the added

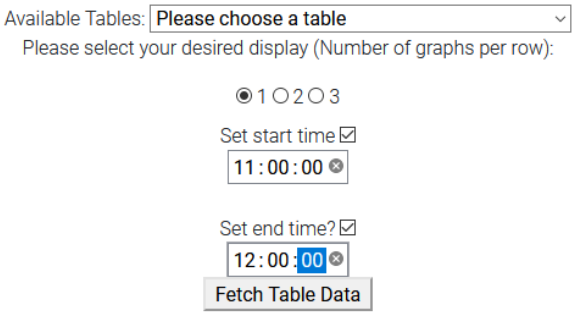


Figure 4.12: "History" Page Additional Customization Features.

specificity. There are 2 main customization features used by the "History" page, the second of which is exclusive to itself.

The first one of these features is the amount of graphs shown in each row. Since the purpose of the layouts in the "History" page is to convey the data across an interval of time, all the data is shown through graphs and, due to the specificity of each layout, the amount of characteristics that will be shown is known from the get go. As such, it is then possible to change the way the fixed number of graphs are shown in the page in relatively simple fashion, depending on user preference. The options range from 1 to 3 graphs in each row of the page.

The second feature (which is exclusive to this page) is the selection of the interval of data that is displayed based on time. The user is capable of choosing whether a start and/or end time is set for the data display, hence the importance of showing the time intervals to the user when selecting tables. Again, given that the source of data and overall structure of the table in which it is stored is known, all the characteristics are associated with the column relating to time and, as such, the shown data intervals are possible to limit based on the time they were acquired.

The 2 previously mentioned customization features are represented in Figure 4.12.

Lastly, and similarly to the "Dashboard" page, there is a world map

display that is used when geographical data is provided. What differs this application of the world map from that of the "Dashboard" page is that all the geographical positions that were stored are represented and have their respective markers associated with the time when the positional data was taken. The data time interval customization described previously is also applied to the world map, meaning that the positions displayed on the map themselves can be filtered by the time they were acquired on.

It is important to note that the features shown for the history page are applied to the specific case of the developed autonomous system preset visualization layout. Other layouts, be it developed or to be developed, may or may not have the same overall visualization options and features.

4.3.5 World Map Display

All the data visualization pages can include some sort of world map positioning display if applicable, as seen in the previous sub sections. This incorporation utilizes an open-source, *JavaScript* solution by the name of "Leaflet", which allows for the representation of the world map in a multitude of different ways [67]. "Leaflet" itself is a contributor to the "OpenStreetMap" project, which consists in a map of the world, free to use in any way under an open license [68].

Besides visual representation of the world map, "Leaflet" also allows for the pinning of markers and drawing of shapes in the map itself wherever the geographical coordinates provided dictate, being that these coordinates depend on the settings set by the user in the case of the dashboard page, and the name of the columns in the case of the history page.

As mentioned previously, the world map representation is customizable and such is done through the use of different "map tiles" provided by other "OpenStreetMap" contributors. In this case, the chosen map tiles are those provided by the company "MapTiler", which provides map de-

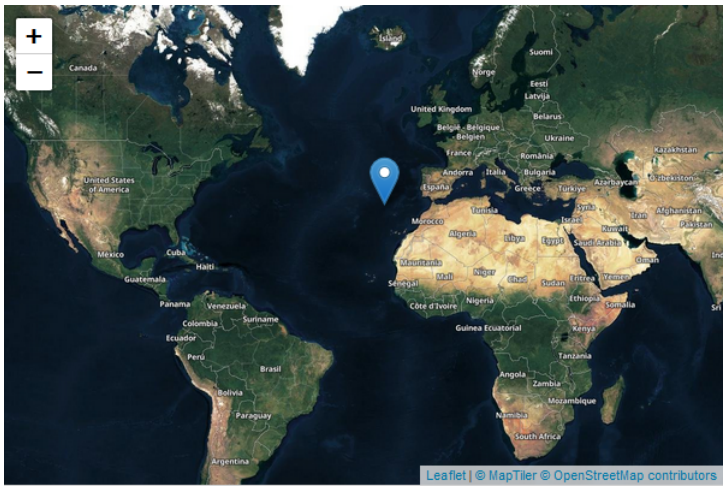


Figure 4.13: World Map Example.

signs for both it's own desktop/mobile solutions and cloud-based services that utilize "OpenStreetMap"[69].

The base world map, used in multiple pages of the solution as described, is represented in Figure 4.13, including a marker defined in example coordinates.

4.3.6 Additional Pages

Lastly, it is important to note that, given the nature of how the overall solution was developed, adding user-developed pages to it is fairly easy to do if necessary, both when it comes to adding visualization presets to the history page and any additional page that is seen as befitting to be added to the visualization tool by said user.

Once a page to be added is developed, then the user can add it to the server-side web "Online Data Viewer" folder so it can then be accessed by the existing pages of the solution.

If the user wishes to add that page as a new visualization preset to the "History" page, all that is necessary is to provide a clickable link inside the body of the "history.html" file to the developed page, making it possible to

access the new preset through the existing “Online Data Viewer” solution.

On the other hand, if the user decides to add a completely new page to the overall solution, it is only necessary to allocate a space for its link in the “navbar” that exists in every page of the “Online Data Viewer”. Adding the new page to the entire solution may take some time, but it is a relatively simple procedure.

This page was intentionally left blank.

Chapter 5

Results

The final outputs of the described general solution will be demonstrated in this chapter. This includes the outputs of the "Database Tool", as well as the final design and functionalities of the "Online Data Viewer".

5.1 Database Tool

In this section, the final design and functionalities of the "Database Tool" will be shown. This will be mostly done through table configuration examples and database table demonstrations.

5.1.1 Log-File Table Creation

When it comes to results regarding the "Database Tool", firstly some of the tables and characteristics of said tables will be demonstrated. In Figure 5.1 we can observe the raw data from a log-file which contain 12 columns, each with a different characteristic specified by an header that is printed every few cycles of data output by the autonomous system responsible for it. These characteristics are the time stamp (defined by its value at the time of header print), followed by all the data acquired through the sensor interface of the autonomous system.

```
1584320400.991782    3.24 18.496 46.568 35.147  102.18    7.75  8.192   0.23   0.57  -0.22 12:31:54.56M+
1584320401.999759    3.22 18.502 46.572 35.146  102.18    7.75  8.193   0.24   0.71  -0.25 12:31:55.67M+
1584320403.055841    3.18 18.499 46.568 35.145  102.18    7.75  8.192   0.10   0.61  -0.25 12:31:56.63M+
1584320404.063843    3.16 18.498 46.568 35.145  102.18    7.75  8.192  -0.14   0.64  -0.47 12:31:57.74M+
1584320405.103873    3.27 18.500 46.569 35.145  102.21    7.75  8.193   0.02   0.58  -0.29 12:31:58.69M+
1584320406.127886
1584320406.128013 Keyb.Cmd: <ESC>Leave data acquisition
1584320406.144034 Press  Temp   Cond   Sal    O_02%   O_02ppm  pH    Chl(a) Tur(FTU) PThrin Time&Memory
1584320406.160275    3.42 18.498 46.569 35.146  102.21    7.75  8.192   0.15   0.74  -0.11 12:31:59.79M+
1584320407.167921    3.47 18.500 46.570 35.145  102.23    7.75  8.192   0.14   0.66  -0.27 12:32:00.74M+
1584320408.191973    3.52 18.498 46.568 35.145  102.23    7.75  8.192   0.17   0.60  -0.12 12:32:01.86M+
1584320409.231984    3.52 18.498 46.567 35.145  102.27    7.76  8.192  -0.10   0.65  -0.19 12:32:02.81M+
1584320410.256017    3.50 18.494 46.566 35.147  102.27    7.76  8.192  -0.04   0.52  -0.33 12:32:03.92M+
1584320411.280031    3.54 18.499 46.571 35.147  102.17    7.75  8.195  -0.09   0.66  -0.52 12:32:04.87M+
1584320412.304092    3.65 18.502 46.574 35.147  102.17    7.75  8.192   0.19   0.57  -0.40 12:32:05.97M+
1584320413.344115    3.57 18.505 46.575 35.146  102.24    7.75  8.192  -0.01   0.56  -0.17 12:32:06.92M+
1584320414.368131    3.47 18.504 46.575 35.146  102.24    7.75  8.192  -0.10   0.56  -0.42 12:32:08.03M+
1584320415.408149    3.40 18.504 46.574 35.145  102.24    7.75  8.192  -0.11   0.68  -0.25 12:32:08.98M+
1584320416.432216    3.45 18.504 46.571 35.143  102.24    7.75  8.192  -0.17   0.68  -0.23 12:32:10.09M+
1584320417.472217    3.59 18.503 46.573 35.145  102.18    7.75  8.192   0.09   0.61  -0.16 12:32:11.04M+
1584320418.480249    3.71 18.502 46.573 35.146  102.18    7.75  8.192   0.08   0.60  -0.45 12:32:12.15M+
1584320419.520271    3.76 18.501 46.573 35.147  102.24    7.75  8.192   0.16   0.77  -0.23 12:32:13.10M+
1584320420.544305    3.71 18.499 46.566 35.143  102.24    7.75  8.192  -0.19   0.62  -0.19 12:32:14.21M+
1584320421.584334    3.47 18.495 46.567 35.147  102.17    7.75  8.192   0.30   0.62  -0.51 12:32:15.16M+
1584320422.608361    3.26 18.498 46.567 35.145  102.17    7.75  8.193  -0.01   0.70  -0.22 12:32:16.27M+
1584320423.632375    3.17 18.496 46.566 35.146  102.20    7.75  8.193   0.11   0.61  -0.19 12:32:17.22M+
1584320424.656416    3.20 18.496 46.567 35.147  102.20    7.75  8.192   0.07   0.64  -0.29 12:32:18.33M+
1584320425.696449    3.24 18.497 46.568 35.146  102.15    7.75  8.193   0.02   0.60  -0.28 12:32:19.28M+
1584320426.720485    3.21 18.499 46.568 35.144  102.15    7.75  8.192   0.12   0.69  -0.38 12:32:20.38M+
1584320427.760543    2.88 18.499 46.568 35.145  102.17    7.75  8.193   0.20   0.62  -0.14 12:32:21.33M+
1584320428.768550    2.64 18.497 46.568 35.147  102.17    7.75  8.193   0.01   0.44  -0.46 12:32:22.44M+
1584320429.824557    2.68 18.495 46.567 35.147  102.14    7.75  8.194   0.14   0.70  -0.11 12:32:23.40M+
1584320430.832604
1584320430.832737 Keyb.Cmd: <ESC>Leave data acquisition
```

Figure 5.1: Raw Log-File Data Snippet.

In Figure 5.2, on the other hand, we can observe the data acquired from said log after it was put through the developed "Database Tool" into a table inside a database, which was visualized through "HeidiSQL"[2].

As we can see, the header of the generated table is the same as the very first one printed in the file, and all the repetitions of that header have been skimmed over, with the exception of the last 2 columns ("Date" and "Time"), which were generated due to the time stamp conversion mechanisms that were utilized in this scenario. Adding to that, all the lines inside the log-file that were either empty or had "clutter" data were ignored, creating a table with a single, continuous flow of information, all of it generated automatically.

When it comes to the data types of each column, as seen in Figure 5.3, the time stamp and "TimeMemory" columns were defined as their values

[2]Since the "MariaDB" default work environment isn't ideal when it comes to demonstrating visually appealing tables, an external tool called "HeidiSQL" was used, in order to visualize how the configuration of tables and databases was set at the end of development[70].

1584320406.144034	Press	Temp	Cond	Sal	O_O2%	O_O2ppm	pH	Chl(a)	Tur(FTU)	PThrin	Time&Memory	Date	Time
1 584 320 400,9917820	3,24	18,496	46,568	35,147	102,18	7,75	8,192	0,23	0,57	-0,22	12:31:54.56M+	2020-03-16	01:00:00
1 584 320 401,9997590	3,22	18,502	46,572	35,146	102,18	7,75	8,193	0,24	0,71	-0,25	12:31:55.67M+	2020-03-16	01:00:01
1 584 320 403,0558410	3,18	18,499	46,568	35,145	102,18	7,75	8,192	0,1	0,61	-0,25	12:31:56.63M+	2020-03-16	01:00:03
1 584 320 404,0638430	3,16	18,498	46,568	35,145	102,18	7,75	8,192	-0,14	0,64	-0,47	12:31:57.74M+	2020-03-16	01:00:04
1 584 320 405,1038730	3,27	18,5	46,569	35,145	102,21	7,75	8,193	0,02	0,58	-0,29	12:31:58.69M+	2020-03-16	01:00:05
1 584 320 406,1602750	3,42	18,498	46,569	35,146	102,21	7,75	8,192	0,15	0,74	-0,11	12:31:59.79M+	2020-03-16	01:00:06
1 584 320 407,1679210	3,47	18,5	46,57	35,145	102,23	7,75	8,192	0,14	0,66	-0,27	12:32:00.74M+	2020-03-16	01:00:07
1 584 320 408,1919730	3,52	18,498	46,568	35,145	102,23	7,75	8,192	0,17	0,6	-0,12	12:32:01.86M+	2020-03-16	01:00:08
1 584 320 409,2319840	3,52	18,498	46,567	35,145	102,27	7,76	8,192	-0,1	0,65	-0,19	12:32:02.81M+	2020-03-16	01:00:09
1 584 320 410,2560170	3,5	18,494	46,566	35,147	102,27	7,76	8,192	-0,04	0,52	-0,33	12:32:03.92M+	2020-03-16	01:00:10
1 584 320 411,2800310	3,54	18,499	46,571	35,147	102,17	7,75	8,195	-0,09	0,66	-0,52	12:32:04.87M+	2020-03-16	01:00:11
1 584 320 412,3040920	3,65	18,502	46,574	35,147	102,17	7,75	8,192	0,19	0,57	-0,4	12:32:05.97M+	2020-03-16	01:00:12
1 584 320 413,3441150	3,57	18,505	46,575	35,146	102,24	7,75	8,192	-0,01	0,56	-0,17	12:32:06.92M+	2020-03-16	01:00:13
1 584 320 414,3681310	3,47	18,504	46,575	35,146	102,24	7,75	8,192	-0,1	0,56	-0,42	12:32:08.03M+	2020-03-16	01:00:14
1 584 320 415,4081490	3,4	18,504	46,574	35,145	102,24	7,75	8,192	-0,11	0,68	-0,25	12:32:08.98M+	2020-03-16	01:00:15
1 584 320 416,4322160	3,45	18,504	46,571	35,143	102,24	7,75	8,192	-0,17	0,68	-0,23	12:32:10.09M+	2020-03-16	01:00:16
1 584 320 417,4722170	3,59	18,503	46,573	35,145	102,18	7,75	8,192	0,09	0,61	-0,16	12:32:11.04M+	2020-03-16	01:00:17
1 584 320 418,4802490	3,71	18,502	46,573	35,146	102,18	7,75	8,192	0,08	0,6	-0,45	12:32:12.15M+	2020-03-16	01:00:18
1 584 320 419,5202710	3,76	18,501	46,573	35,147	102,24	7,75	8,192	0,16	0,77	-0,23	12:32:13.10M+	2020-03-16	01:00:19
1 584 320 420,5443050	3,71	18,499	46,566	35,143	102,24	7,75	8,192	-0,19	0,62	-0,19	12:32:14.21M+	2020-03-16	01:00:20
1 584 320 421,5843340	3,47	18,495	46,567	35,147	102,17	7,75	8,192	0,3	0,62	-0,51	12:32:15.16M+	2020-03-16	01:00:21
1 584 320 422,6083610	3,26	18,498	46,567	35,145	102,17	7,75	8,193	-0,01	0,7	-0,22	12:32:16.27M+	2020-03-16	01:00:22

Figure 5.2: Output Log-File Table Values Snippet. Visualized through "HeidiSQL".

demanded by the user, "decimal" and "char" respectively, whilst the final 2 columns of the table, which include converted date and time data, were defined as they are. The remaining columns were defined as floats.

5.1.2 Real-Time Table Creation

In order to test the real-time table creation and since the data relating to practical applications available for testing was stored in log-files, the socket communication features needed to be tested in some shape or form. To do this, a small *Python* program simulating a socket client was created inside the same machine that ran the "Database Tool", so that the client/socket interactions could be tested.

The functionality of the developed socket client is that of sending simulated lines of sensor data, testing the data acquisition on the side of the socket server. The data was sent as full lines of separated values over 2000 row increments, and included a total of 9 different columns, some of these including simulated sensor outputs and others including data for mechanism testing purposes.

In order to test the table header acquisition features in real-time, an

1	1584320406.14...	DECIMAL
2	Press	FLOAT
3	Temp	FLOAT
4	Cond	FLOAT
5	Sal	FLOAT
6	O_O2%	FLOAT
7	O_O2ppm	FLOAT
8	pH	FLOAT
9	Chl(a)	FLOAT
10	Tur(FTU)	FLOAT
11	PThrin	FLOAT
12	Time&Memory	VARCHAR
13	Date	DATE
14	Time	TIME

Figure 5.3: Output Log-File Table Header Characteristics. Visualized through "HeidiSQL".

additional feature was added to the developed socket client that sends a table header in the same way that the value lines of data were.

Please note that the database in which the data was stored in this scenario was different than the database seen in the log-file table creation scenario, given that the data was manufactured and used solely for testing purposes unlike the log-file table creation example.

In Figure 5.4 the data output from the "Database Tool" to a table inside a database is represented. As we can see, the data columns sent in real-time via socket communication vary significantly. The first column, "SimLine", includes a counter of the number of lines, inclusively, sent so far in real time. The second column,"SimTimestamp", includes a time stamp value to be used in order to test the time stamp date/time conversion mechanisms. The third through ninth columns, "SimData1" through "SimData7", include simulated sensor data, which randomly sets a different value inside a threshold for each of the columns every time a new line is sent. Finally, the last 2 columns, "Date" and "Time", were not a part of the lines of data sent in real-time, but are instead a result of the time stamp conversion of the value seen in the "SimTimestamp" column.

SimLine	SimTimestamp	SimData1	SimData2	SimData3	SimData4	SimData5	SimData6	SimData7	Date	Time
0	1 584 356 546,7005470	4	18	12	10	5	4	4	2020-03-16	11:02:26
1	1 584 356 546,7005470	5	6	28	12	2	8	4	2020-03-16	11:02:26
2	1 584 356 546,7005470	9	21	35	14	12	5	0	2020-03-16	11:02:26
3	1 584 356 546,7005470	0	1	7	14	2	2	4	2020-03-16	11:02:26
4	1 584 356 546,7005470	6	24	7	22	8	0	1	2020-03-16	11:02:26
5	1 584 356 546,7005470	9	8	23	29	12	2	4	2020-03-16	11:02:26
6	1 584 356 546,7005470	6	5	7	18	11	0	1	2020-03-16	11:02:26
7	1 584 356 546,7005470	7	14	24	5	6	1	3	2020-03-16	11:02:26
8	1 584 356 546,7005470	1	8	4	12	12	3	1	2020-03-16	11:02:26
9	1 584 356 546,7005470	4	4	12	13	9	1	0	2020-03-16	11:02:26
10	1 584 356 546,7005470	3	3	9	6	5	1	4	2020-03-16	11:02:26
11	1 584 356 546,7005470	2	22	9	6	9	0	1	2020-03-16	11:02:26
12	1 584 356 546,7005470	7	7	26	24	13	3	2	2020-03-16	11:02:26
13	1 584 356 546,7005470	0	1	20	7	2	3	2	2020-03-16	11:02:26
14	1 584 356 546,7005470	1	18	8	3	12	6	0	2020-03-16	11:02:26
15	1 584 356 546,7005470	5	9	20	12	5	3	2	2020-03-16	11:02:26
16	1 584 356 546,7005470	7	3	32	28	0	9	1	2020-03-16	11:02:26
17	1 584 356 546,7005470	5	23	3	21	4	3	2	2020-03-16	11:02:26
18	1 584 356 546,7005470	6	8	38	2	8	7	3	2020-03-16	11:02:26
19	1 584 356 546,7005470	6	16	36	9	13	9	3	2020-03-16	11:02:26
20	1 584 356 546,7005470	3	21	13	12	5	4	3	2020-03-16	11:02:26
21	1 584 356 546,7005470	8	7	3	21	13	7	2	2020-03-16	11:02:26

Figure 5.4: Real-Time Simulation Output Table Values Snippet. Visualized through “HeidiSQL”.

When it comes to the characteristics of the header shown in Figure 5.4, they are represented in Figure 5.5. As we can see by analyzing the Figure, the “SimTimestamp” was defined through the “Database Tool” as a “decimal” column, given that it includes a time stamp, whilst the “Date” and “Time” columns were defined as their names imply. The remaining columns were then defined as “floats”.

1	SimLine	FLOAT
2	SimTimestamp	DECIMAL
3	SimData1	FLOAT
4	SimData2	FLOAT
5	SimData3	FLOAT
6	SimData4	FLOAT
7	SimData5	FLOAT
8	SimData6	FLOAT
9	SimData7	FLOAT
10	Date	DATE
11	Time	TIME

Figure 5.5: Real-Time Simulation Output Table Header Characteristics. Visualized through “HeidiSQL”.

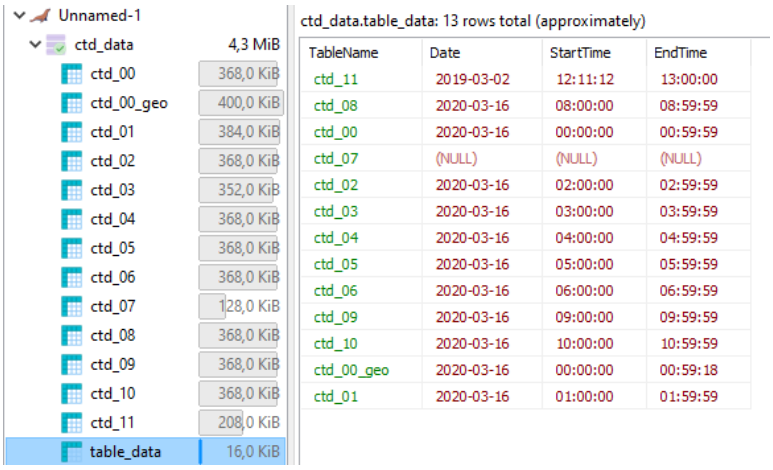


Figure 5.6: Database Date/Time Configuration. Visualized through "HeidiSQL".

5.1.3 Additional Date/Time Configurations

When it comes to the date/time configuration mechanisms relating to the "table_data" table, an example of it is represented in Figure 5.6. The date/time configurations of the tables shown are meant as an example of what shape they would normally take, depending on how the configuration itself is carried out.

Analyzing the aforementioned Figure 5.6, we can observe that the data relating to most of the table names were all taken in the same day at different times, date/time which was acquired through conversion of the time stamps inside the log-files for each of these tables. The table "ctd_11", on other hand, seems to have been taken on a completely different day/hour. This isn't necessarily the result of the conversion of time stamps inside the utilized log-file, but instead a date and time inserted manually by the user for testing purposes.

The "ctd_07" table name, on the other hand, does not have any date and time data associated with it for the testing purposes mentioned previously, being defined as such by user input.

Lastly, it is important to note that, if we compare Figure 5.2 with Figure

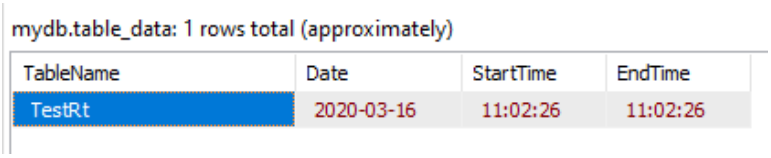


Figure 5.7: Database Real-Time Date/Time Configuration. Visualized through "HeidiSQL".

5.6, the "Time&Memory" column in the "ctd_01" table isn't correlated directly to the time observed in the "table_data" columns regarding the start and end time of the log. This is due to the fact that the "Time&Memory" column in the utilized log-files was taken in the context of a different time-zone than that of the time stamps in that same log and, as such, is nothing but an additionally observed variable in practical terms.

The previously shown results are based on the log-file data implementation, however when it comes to the real-time data implementation, the date/time configuration can be seen in Figure 5.7. As mentioned, this implementation was done in a different database, given that it was based on simulated data and not real data. When it comes to the date/time configuration itself, the values were acquired based upon the time and date the "Database Tool" acquired from the operating system of the machine that ran it, as the process of real-time database table writing was being executed.

5.2 Online Data Viewer

In this section, the final version of the "Online Data Viewer" will be demonstrated. This demonstration will focus primarily on the final designs and functionalities of the pages that comprise it.

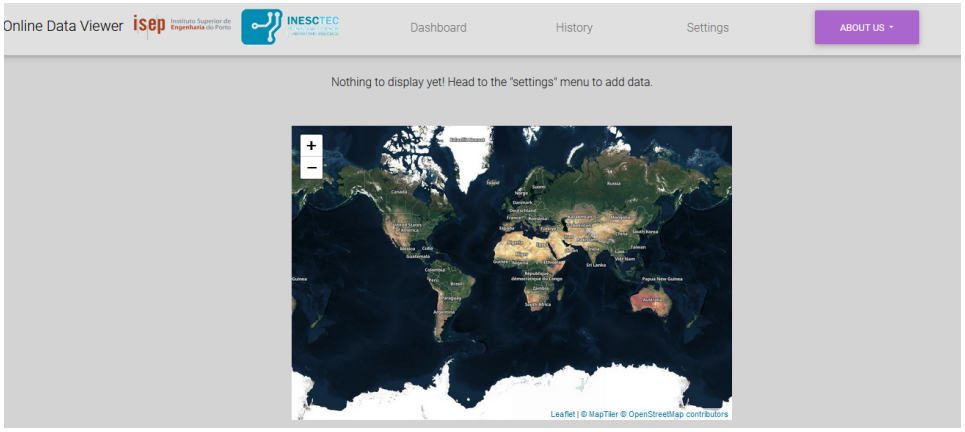


Figure 5.8: "Dashboard" Page Shown to the User Prior to Any Configuration.

5.2.1 Dashboard & Settings

Dashboard

The "Dashboard" page of the final online data viewer is shown to the user in the manner represented in Figure 5.8, the very first time it is opened after the solution has been installed.

The page shown in Figure 5.8 is a result of no visualization settings being defined yet for the "Dashboard" page, ever since the solution was installed into a server. If any user had added a visualization setting through the "Settings" page since, this page would not be shown in this fashion and, instead, the last visualization setting configured would.

That being said, and has explained throughout this thesis, there are a few visualization options available to the user for the configuration of the "Dashboard" page. The first of these visualization options are the gauges (both linear and radial) and text boxes, which are shown in their final form through the configured "Dashboard" page in Figure 5.9. In it, an example of both types of gauges is shown, along with an example of a text box display.

In order to demonstrate the measurement unit specification feature and as we can see in Figure 5.9, the "Temp"(relating to temperature), "Press"

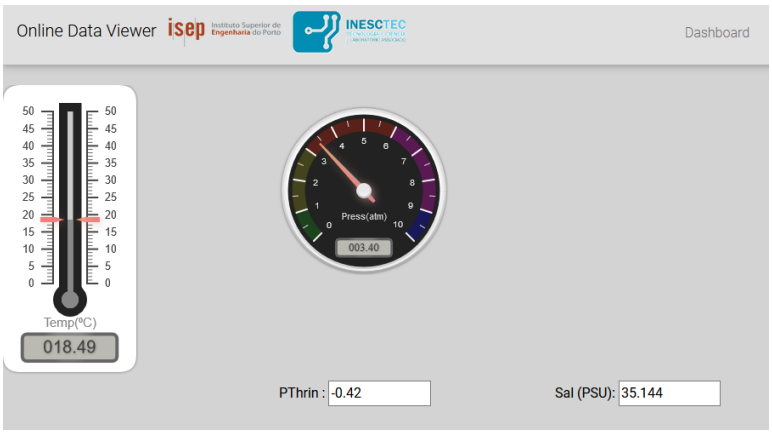


Figure 5.9: Display of the 3 Fundamental Final Dashboard Visualization Options.

(relating to pressure) and "Sal" (relating to salinity levels) data representation options were associated with degrees ("^oC"), atmospheres ("atm") and Practical Salinity Units ("PSU") respectively.

Further down in the same page, 2 graphs were configured which share the same variables in both the "X" an "Y" axis, with a different configured number of latest data entries for each one and represented with one graph per row. This was done to demonstrate both the graphs themselves and how the number of configured entries affects the data display of said graphs. In Figure 5.10 the first graph is represented, configured to show the 50 latest table data entries, and in Figure 5.11 the second graph is represented, which is configured to have 500 of the latest table data entries.

As an example, a second "Dashboard" page was setup so that the configurable number of graphs per row could be demonstrated. In Figure 5.12, 4 different graphs can be seen, being that 2 graphs are configured per row. The graphs in the same rows have the same variables but different number of entries (graphs on the left hold 10 of the latest entries and the ones on the right hold 100 of the latest entries).

Returning to the main configured "Dashboard" page, a comparison graph was setup between 3 different variables, over the last 150 fetched data en-

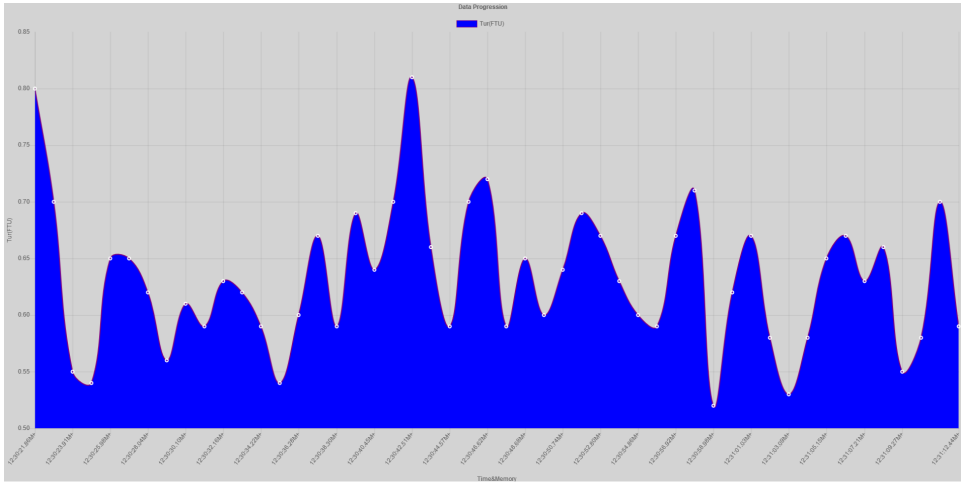


Figure 5.10: Final Dashboard Graph Display (50 Latest Entries).

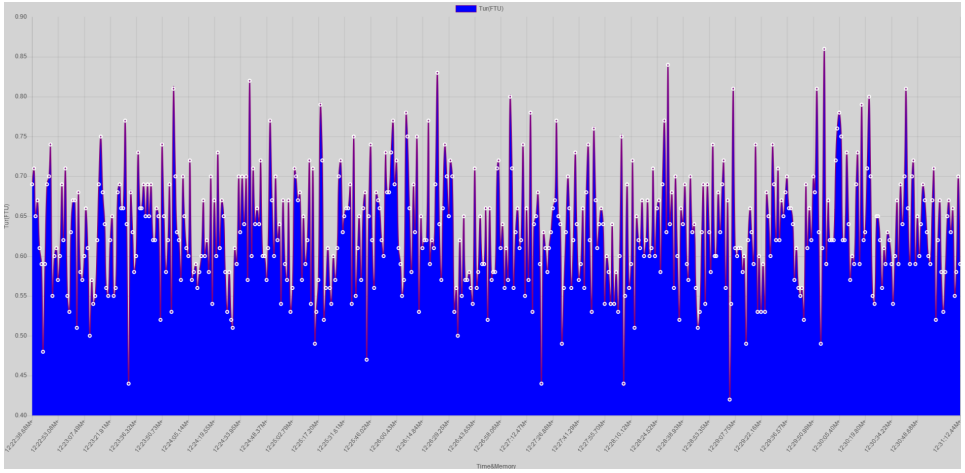


Figure 5.11: Final Dashboard Graph Display (500 Latest Entries).

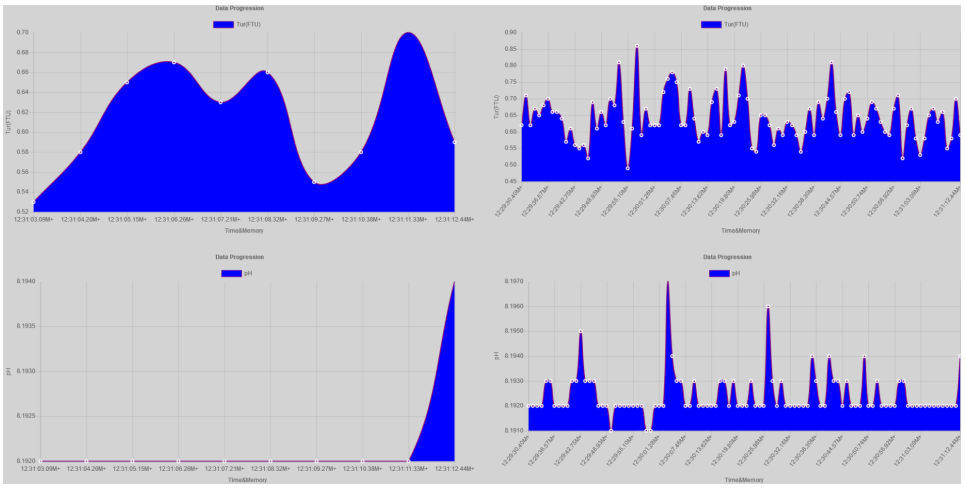


Figure 5.12: Graph Display Example (4 Graphs, 2 per Row).

tries. This comparison graph can be seen in Figure 5.13.

Lastly, and returning to the main configured “Dashboard” page, a world map display was added. This world map is represented in Figure 5.14 and utilizes the last known geographical position inside the provided table data column. The marker seen in the world map was configured using the “Fetch Current Time From System” selection, which, like its namesake implies, utilizes the current system time of the server that it is running on for the marker display.

Settings

In order to demonstrate the functionalities of the “Settings” page, the configuration process for the previously described “Dashboard” page will be shown, along with the table from which the data was fetched.

Firstly, the “Settings” page portion relating to the initial setup of the RDBMS connection parameters can be seen in Figure 5.15. Given that the RDBMS was being ran inside the machine, the shown host address is simply the “localhost”.

When inside the main portion of the “Settings” page, the database and

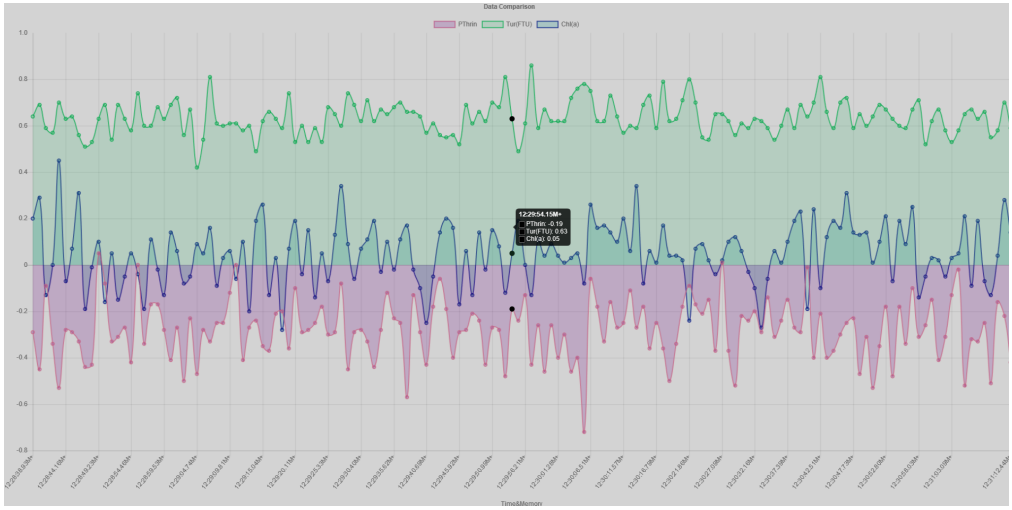


Figure 5.13: Final Dashboard Comparison Graph.

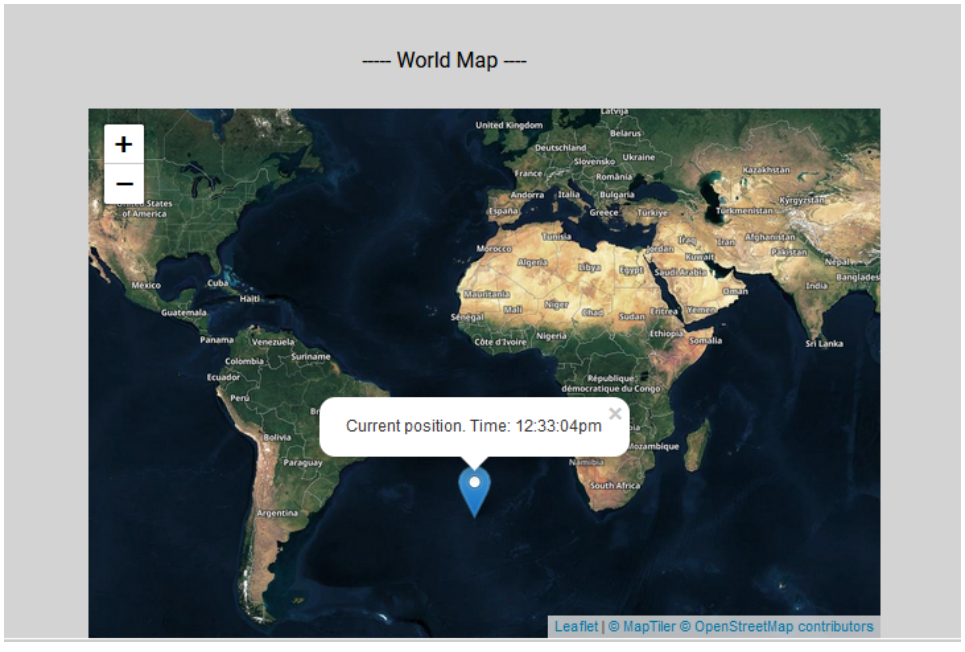


Figure 5.14: Final Dashboard World Map Display (With Current Time Stamp Marker).

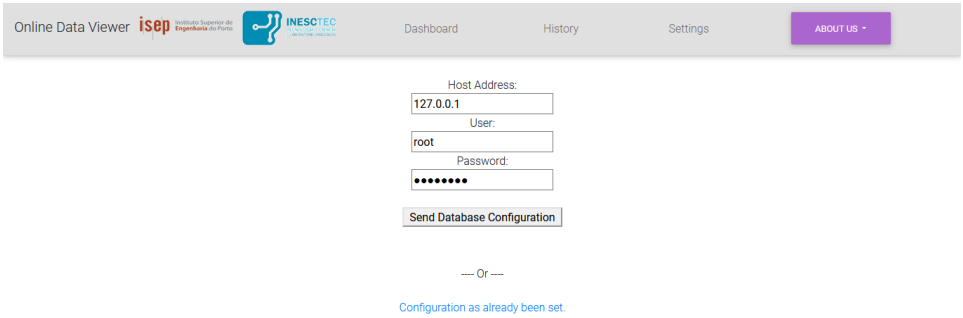


Figure 5.15: Initial Database Connection Parameters.

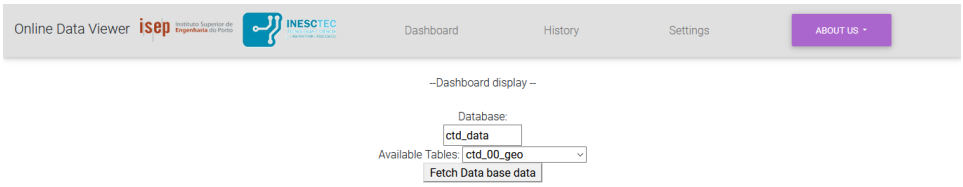


Figure 5.16: Dashboard Configuration Table and Database Selection.

table from which the data should be taken from were selected, them being the "ctd_data" database and "ctd_00_geo" table as seen in Figure 5.16.

From there on out, the selected database and table were submitted, resulting in the acquisition of the selected table column names.

In Figure 5.17 the options selected for the "Dashboard" page's gauge display is shown. In Figure 5.18, on the other hand, the options relating to the text boxes shown in the "Dashboard" page are represented.

When it comes to the graphs, the options selected for the main page are represented in Figure 5.19, whilst the options for the graphs that weren't used along with the configuration of the number of graphs per row can be seen in Figure 5.20. Analyzing the figure, we can see that both graphs include the same variables but a different number of latest entries to show, which results in the graphs seen in Figures 5.10 and 5.11.

As for the graph settings associated with the second "Dashboard" page used to demonstrate the "graph per row" configuration options, these are

Figure 5.17: Dashboard Configuration Gauge Selection.

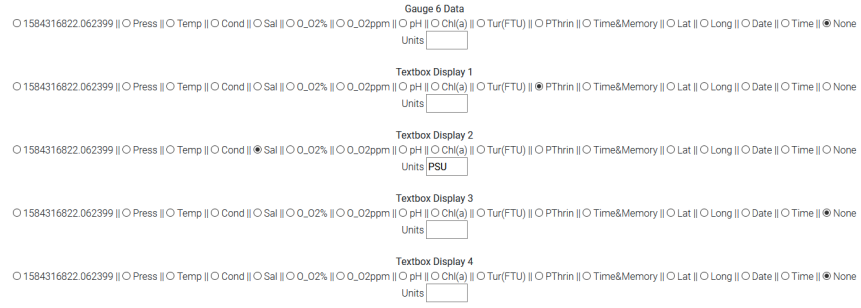


Figure 5.18: Dashboard Configuration Text Box Selection.

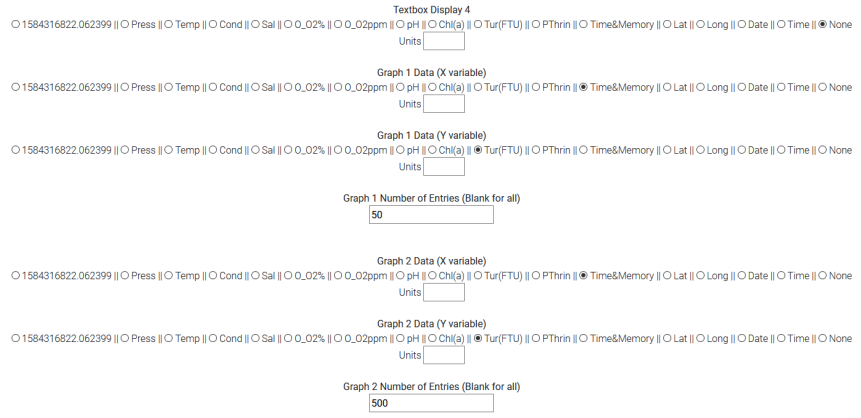


Figure 5.19: Main Dashboard Configuration Graph Selection.

Figure 5.20: Main Dashboard Configuration Graph Selection (Unused Graphs + Number of Graphs per Row).

represented in Figures 5.21 and 5.22.

Returning to the main "Dashboard" page, the options selected for the comparison graph representation can be seen in Figure 5.23.

As we can see up to this point, every single variable had its own field which allowed its users to specify the measurement unit associated with each of the data variables that were going to be represented in the "Dashboard" page.

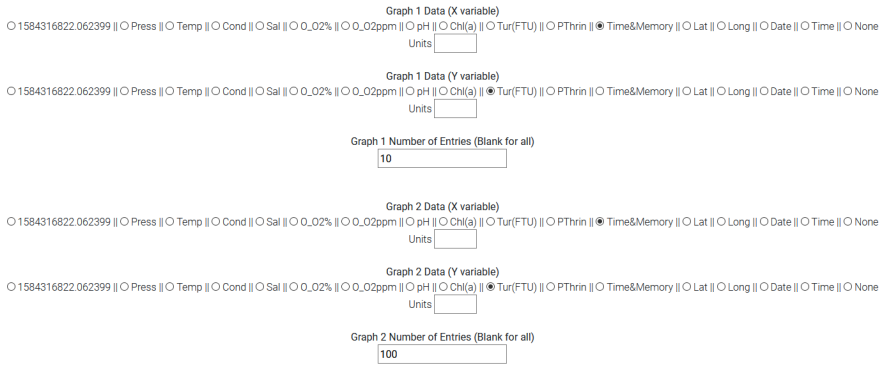


Figure 5.21: Example Dashboard Configuration Graph Selection (First Row of Graphs).

Figure 5.22: Example Dashboard Configuration Graph Selection (Second Row of Graphs + Number of Graphs per Row).

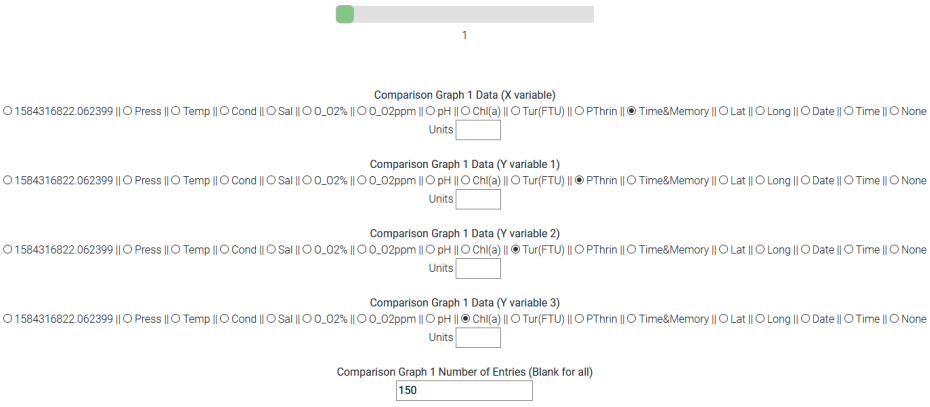


Figure 5.23: Main Dashboard Configuration Comparison Graph Selection.

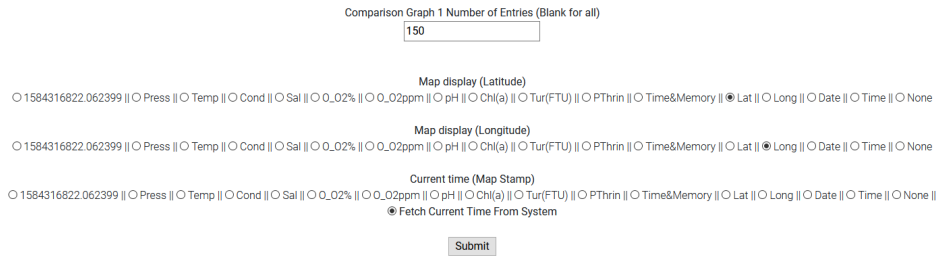


Figure 5.24: Dashboard Configuration Map Selection.

Finally, the options selected for the world map display at the end of the main "Dashboard" page are represented in Figure 5.24. The latitude and longitude were set for the marker, and its contents were set based on the current time from the server that ran the "Online Data Viewer".

Data Used

The data that was utilized for the visualization example of the "Dashboard" and "Settings" pages was taken from the table represented in Figure 5.25. The values seen here are the final values of the table itself, and as such the values in the gauges and text boxes included in the "Dashboard" page will be those seen in the final row of Figure 5.25. As mentioned previously, the graphs utilize the configured number of graph entries to show that amount of final table values. For example, if an user configures the number of graph entries to be 30 whilst acquiring the data from a table with 400 rows, the solution will utilize the data from row 371 all the way to row 400, inclusively.

It is important to note that, originally, the log-file utilized for testing did not have any geographical data associated to it, given that it was included in a different file. Both log-files were combined utilizing a simple *Python* program developed specifically for this purpose, whilst also cleaning some the unnecessary lines of data for good measure.

This resulted in the log-file that was utilized to generate the table seen in the previously mentioned Figure 5.25, whose snippet is represented in

1584316822.062399	Press	Temp	Cond	Sal	O_O2%	O_O2ppm	pH	Chl(a)	Tur(FTU)	PThrin	Time&Memory	Lat	Long	Date	Time
1 584 320 337,1660120	2,84	18,503	46,572	35,145	102,34	7,76	8,192	0,01	0,64	-0,53	12:30:50.74M+	-36,4035	-12,6769	2020-03-16	00:58:57
1 584 320 338,1900630	2,52	18,502	46,57	35,145	102,34	7,76	8,192	0,1	0,69	-0,35	12:30:51.85M+	-36,4035	-12,677	2020-03-16	00:58:58
1 584 320 339,2300910	2,43	18,501	46,568	35,144	102,3	7,76	8,193	0,21	0,67	-0,18	12:30:52.80M+	-36,4035	-12,677	2020-03-16	00:58:59
1 584 320 340,2381200	2,34	18,502	46,571	35,145	102,3	7,76	8,192	-0,07	0,63	-0,48	12:30:53.91M+	-36,4035	-12,6771	2020-03-16	00:59:00
1 584 320 341,2781660	2,45	18,5	46,569	35,145	102,27	7,76	8,192	0,19	0,6	-0,18	12:30:54.86M+	-36,4035	-12,6771	2020-03-16	00:59:01
1 584 320 342,3021730	2,78	18,501	46,569	35,144	102,27	7,76	8,192	0,09	0,59	-0,34	12:30:55.97M+	-36,4035	-12,6771	2020-03-16	00:59:02
1 584 320 343,3422050	3,09	18,503	46,57	35,143	102,24	7,75	8,192	0,25	0,67	-0,1	12:30:56.92M+	-36,4035	-12,6772	2020-03-16	00:59:03
1 584 320 344,3661980	3,18	18,504	46,572	35,144	102,24	7,75	8,193	-0,14	0,71	-0,31	12:30:58.03M+	-36,4035	-12,6772	2020-03-16	00:59:04
1 584 320 345,4062420	3,23	18,502	46,569	35,143	102,28	7,76	8,193	-0,05	0,52	-0,26	12:30:58.98M+	-36,4035	-12,6773	2020-03-16	00:59:05
1 584 320 346,4143230	3,42	18,499	46,569	35,146	102,28	7,76	8,192	0,03	0,62	-0,15	12:31:00.08M+	-36,4035	-12,6773	2020-03-16	00:59:06
1 584 320 347,4543740	3,53	18,5	46,567	35,144	102,2	7,75	8,192	0,02	0,67	-0,41	12:31:01.03M+	-36,4035	-12,6774	2020-03-16	00:59:07
1 584 320 348,4783350	3,71	18,501	46,573	35,147	102,2	7,75	8,192	-0,05	0,58	-0,31	12:31:02.14M+	-36,4035	-12,6774	2020-03-16	00:59:08
1 584 320 349,5183970	3,78	18,502	46,573	35,147	102,21	7,75	8,192	0,03	0,53	-0,13	12:31:03.09M+	-36,4035	-12,6775	2020-03-16	00:59:09
1 584 320 350,5423910	3,59	18,505	46,573	35,144	102,21	7,75	8,192	0,05	0,58	-0,02	12:31:04.20M+	-36,4035	-12,6775	2020-03-16	00:59:10
1 584 320 351,5824080	3,59	18,502	46,571	35,144	102,21	7,75	8,192	0,21	0,65	-0,52	12:31:05.15M+	-36,4035	-12,6775	2020-03-16	00:59:11
1 584 320 352,5904990	3,73	18,5	46,57	35,145	102,21	7,75	8,192	-0,09	0,67	-0,32	12:31:06.26M+	-36,4035	-12,6776	2020-03-16	00:59:12
1 584 320 353,6304820	3,88	18,5	46,568	35,143	102,24	7,75	8,192	0,19	0,63	-0,33	12:31:07.21M+	-36,4035	-12,6776	2020-03-16	00:59:13
1 584 320 354,6545110	3,98	18,499	46,57	35,146	102,24	7,75	8,192	-0,07	0,66	-0,25	12:31:08.32M+	-36,4035	-12,6777	2020-03-16	00:59:14
1 584 320 355,6945200	4,05	18,498	46,571	35,147	102,22	7,75	8,192	-0,13	0,55	-0,51	12:31:09.27M+	-36,4035	-12,6777	2020-03-16	00:59:15
1 584 320 356,7507820	3,96	18,496	46,566	35,145	102,22	7,75	8,192	0,04	0,58	-0,16	12:31:10.38M+	-36,4035	-12,6778	2020-03-16	00:59:16
1 584 320 357,7426490	3,7	18,496	46,565	35,145	102,3	7,76	8,192	0,28	0,7	-0,22	12:31:11.33M+	-36,4035	-12,6778	2020-03-16	00:59:17
1 584 320 358,7666100	3,4	18,493	46,562	35,144	102,3	7,76	8,194	0,14	0,59	-0,42	12:31:12.44M+	-36,4036	-12,6779	2020-03-16	00:59:18

Figure 5.25: Final Selected Table Values. Visualized Through "HeidiSQL".

Figure 5.26.

5.2.2 History

The "History" page itself varies greatly depending on the context that it is being applied to. As mentioned in Chapter 4, this page utilizes fixed visualization options depending on the autonomous system that is being observed. In the case of this thesis, two main "History" page was developed, one for the "TowFish" autonomous system and another one for the "FeedFirst" autonomous system.

In Figure 5.27 the initial parameters for the "TowFish" "History" page are represented, in regards to the examples that follow.

As we can see in Figure 5.27, the data was setup to be seen in a 2 graph per row format, with data ranging in time between 00 : 11 : 00 and 00 : 40 : 00.

In this visualization case, a total of 10 different graphs were generated within the guidelines set, 4 of which are represented in Figure 5.28.

Detailing one of the graphs, as seen in Figure 5.29, we can see that the data has been truncated in a way that only includes the specified times in the configuration process.

```
1584320320.637588    2.00 18.497 46.567 35.146  102.12    7.75  8.192  -0.06   0.59  -0.14 12:30:34.22M+ -36.403423653833 -12.676189709000
1584320321.661653    2.13 18.498 46.566 35.145  102.12    7.75  8.192   0.06   0.54  -0.31 12:30:35.32M+ -36.403427833333 -12.676236606667
1584320322.701671    2.28 18.495 46.564 35.145  102.26    7.76  8.192   0.01   0.60  -0.24 12:30:36.28M+ -36.403432560000 -12.676281350333
1584320323.725658    2.45 18.496 46.567 35.147  102.26    7.76  8.194   0.10   0.67  -0.15 12:30:37.39M+ -36.403437975833 -12.676326070667
1584320324.813733    2.58 18.499 46.570 35.147  102.25    7.76  8.193   0.19   0.59  -0.27 12:30:38.30M+ -36.403441566167 -12.676369428833
1584320325.837698    2.75 18.502 46.571 35.144  102.25    7.76  8.192   0.23   0.69  -0.29 12:30:39.50M+ -36.403444045333 -12.676410813333
1584320326.877739    2.71 18.502 46.571 35.145  102.30    7.76  8.192  -0.19   0.64  -0.01 12:30:40.45M+ -36.403444073500 -12.676451202333
1584320327.885760    2.35 18.504 46.574 35.146  102.30    7.76  8.194   0.24   0.70  -0.40 12:30:41.56M+ -36.403443328667 -12.676493286500
1584320328.925784    2.04 18.504 46.573 35.145  102.23    7.75  8.193  -0.10   0.81  -0.21 12:30:42.51M+ -36.403441940667 -12.676537787833
1584320329.949824    1.82 18.502 46.570 35.144  102.23    7.75  8.193   0.12   0.66  -0.40 12:30:43.62M+ -36.403441666167 -12.676582553333
1584320330.989851    1.81 18.499 46.568 35.145  102.26    7.76  8.192   0.19   0.59  -0.37 12:30:44.57M+ -36.403446456333 -12.676627939833
1584320332.014462 Press  Temp   Cond   Sal    O_02%   O_02ppm  pH    Chl(a) Tur(FTU) PThrin Time&Memory Lat Long
1584320332.046142    1.91 18.498 46.567 35.146  102.26    7.76  8.193   0.16   0.70  -0.30 12:30:45.67M+ -36.403459889000 -12.676712756833
1584320333.053952    2.16 18.499 46.570 35.146  102.22    7.75  8.192   0.31   0.72  -0.25 12:30:46.62M+ -36.403467615333 -12.676756424667
1584320334.061932    2.50 18.503 46.573 35.146  102.22    7.75  8.192   0.14   0.59  -0.23 12:30:47.73M+ -36.403473880333 -12.676796909833
1584320335.101967    2.75 18.505 46.575 35.145  102.25    7.76  8.192   0.13   0.65  -0.47 12:30:48.68M+ -36.403476993667 -12.676840287667
1584320336.126056    2.93 18.505 46.574 35.145  102.25    7.76  8.194   0.14   0.60  -0.31 12:30:49.79M+ -36.403475194333 -12.676883276500
1584320337.166012    2.84 18.503 46.572 35.145  102.34    7.76  8.192   0.01   0.64  -0.53 12:30:50.74M+ -36.403474093833 -12.676929028833
1584320338.190063    2.52 18.502 46.570 35.145  102.34    7.76  8.192   0.10   0.69  -0.35 12:30:51.85M+ -36.403476508667 -12.676973714333
1584320339.230091    2.43 18.501 46.568 35.144  102.30    7.76  8.193   0.21   0.67  -0.18 12:30:52.80M+ -36.403480079667 -12.677015700667
1584320340.238120    2.34 18.502 46.571 35.145  102.30    7.76  8.192  -0.07   0.63  -0.48 12:30:53.91M+ -36.403485384167 -12.677058271833
1584320341.278166    2.45 18.500 46.569 35.145  102.27    7.76  8.192   0.19   0.60  -0.18 12:30:54.86M+ -36.403486926667 -12.677100398333
1584320342.302173    2.78 18.501 46.569 35.144  102.27    7.76  8.192   0.09   0.59  -0.34 12:30:55.97M+ -36.403482250000 -12.677145411500
1584320343.342205    3.09 18.503 46.570 35.143  102.24    7.75  8.192   0.25   0.67  -0.10 12:30:56.92M+ -36.403482632667 -12.677187939667
1584320344.366198    3.18 18.504 46.572 35.144  102.24    7.75  8.193  -0.14   0.71  -0.31 12:30:58.03M+ -36.403487868500 -12.677230437333
1584320345.406242    3.23 18.502 46.569 35.143  102.28    7.76  8.193  -0.05   0.52  -0.26 12:30:58.98M+ -36.403498439000 -12.677273461667
1584320346.414323    3.42 18.499 46.569 35.146  102.28    7.76  8.192   0.03   0.62  -0.15 12:31:00.08M+ -36.403508530833 -12.677316817167
1584320347.454374    3.53 18.500 46.567 35.144  102.20    7.75  8.192   0.02   0.67  -0.41 12:31:01.03M+ -36.403514605333 -12.677361245667
1584320348.478335    3.71 18.501 46.573 35.147  102.20    7.75  8.192  -0.05   0.58  -0.31 12:31:02.14M+ -36.403515488500 -12.677406122167
1584320349.518397    3.78 18.502 46.573 35.147  102.21    7.75  8.192   0.03   0.53  -0.13 12:31:03.09M+ -36.403513897833 -12.677451798000
1584320350.542391    3.59 18.505 46.573 35.144  102.21    7.75  8.192   0.05   0.58  -0.02 12:31:04.20M+ -36.403513933000 -12.677496672000
1584320351.582408    3.59 18.502 46.571 35.144  102.21    7.75  8.192   0.21   0.65  -0.52 12:31:05.15M+ -36.403517531833 -12.677540887333
1584320352.590499    3.73 18.500 46.570 35.145  102.21    7.75  8.192  -0.09   0.67  -0.32 12:31:06.26M+ -36.403521728667 -12.677584131000
1584320353.630482    3.88 18.500 46.568 35.143  102.24    7.75  8.192   0.19   0.63  -0.33 12:31:07.21M+ -36.403526406500 -12.677625445000
1584320354.654511    3.98 18.499 46.570 35.146  102.24    7.75  8.192  -0.07   0.66  -0.25 12:31:08.32M+ -36.403530385500 -12.677666151833
1584320355.694520    4.05 18.498 46.571 35.147  102.22    7.75  8.192  -0.13   0.55  -0.51 12:31:09.27M+ -36.403532750667 -12.677706404000
1584320356.719141 Press  Temp   Cond   Sal    O_02%   O_02ppm  pH    Chl(a) Tur(FTU) PThrin Time&Memory Lat Long
1584320356.750782    3.96 18.496 46.566 35.145  102.22    7.75  8.192   0.04   0.58  -0.16 12:31:10.38M+ -36.403542021500 -12.677789292333
1584320357.742649    3.70 18.496 46.565 35.145  102.30    7.76  8.192   0.28   0.70  -0.22 12:31:11.33M+ -36.403548129833 -12.677833289167
1584320358.766610    3.40 18.493 46.562 35.144  102.30    7.76  8.194   0.14   0.59  -0.42 12:31:12.44M+ -36.403554328667 -12.677877096833
```

Figure 5.26: Raw Log-File Data Snippet. Utilized for Testing and Including Geographical Data.

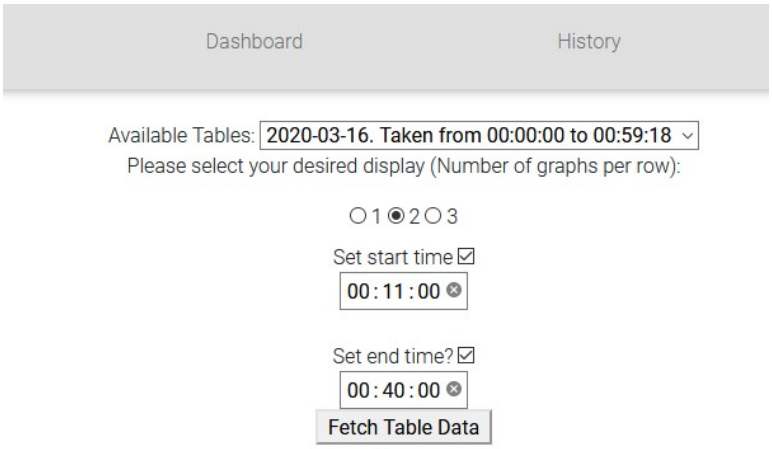


Figure 5.27: Initial "TowFish" Data Visualization Parameters.

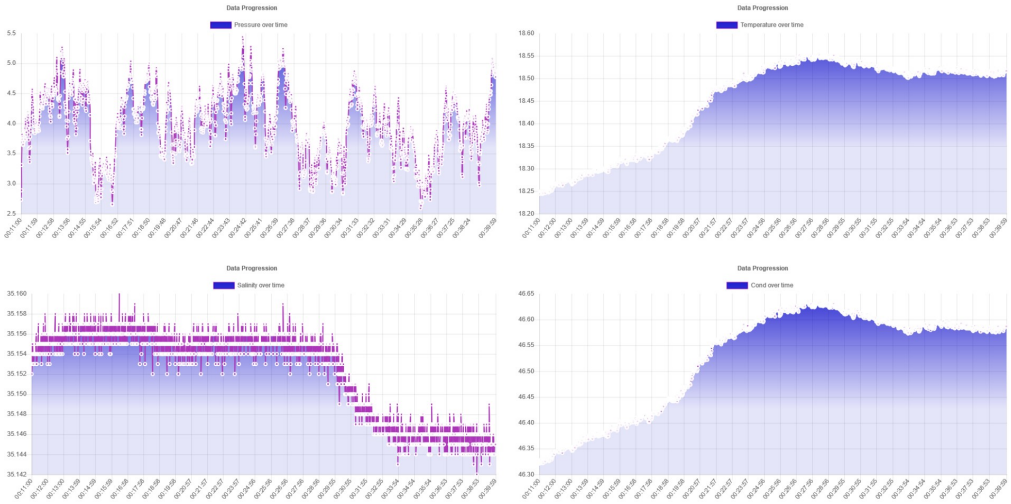


Figure 5.28: "TowFish" Graph Data Visualization.

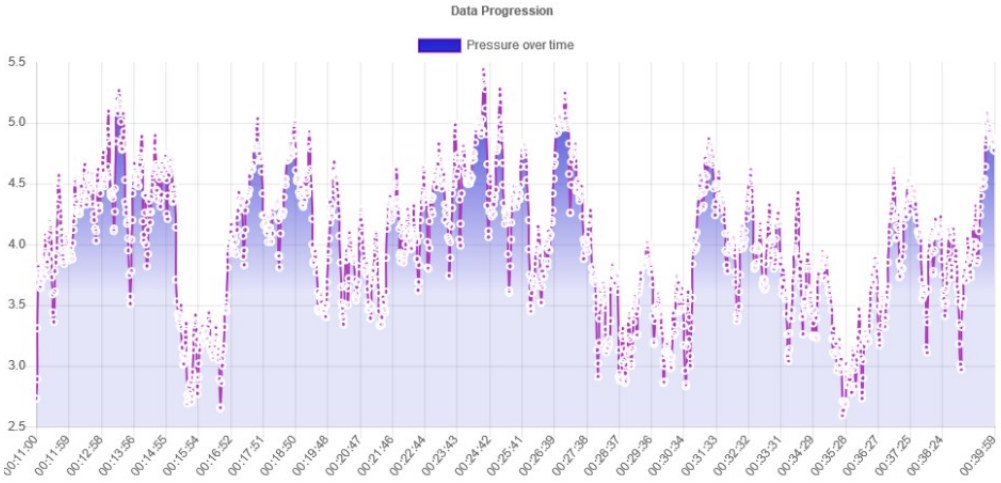


Figure 5.29: "TowFish" Graph Data Visualization (Single Graph).

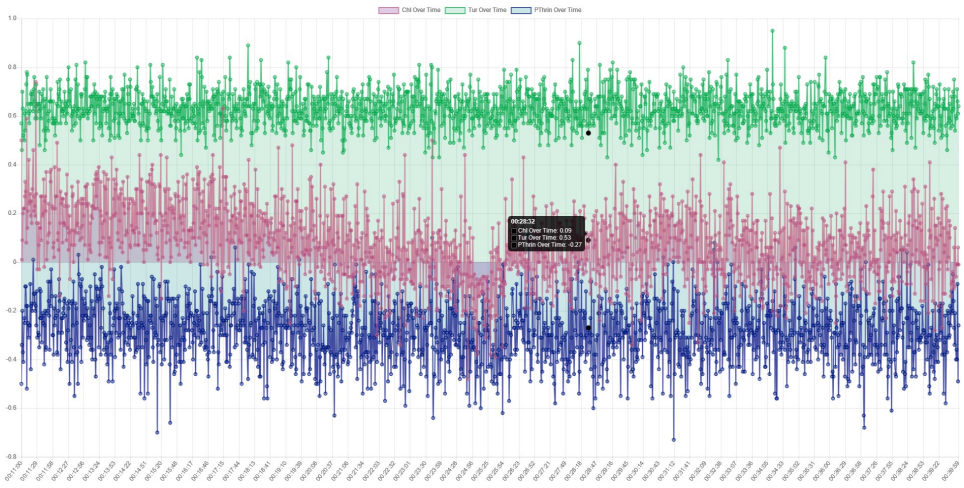


Figure 5.30: “TowFish” Comparison Graph Data Visualization).

One top of these graphs, a comparison graph was added in order to analyze some of the observed variables in regards to each other, as can be seen in Figure 5.30.

Lastly, this “History” page also includes a world map display, which can be seen in Figure 5.31, which also includes an additional note with the state of all the data values at the time that the observed geographical position was taken, that being the last geographical position within the specified time frame.

This map includes markers (represented by red dots) based upon the geographical data present in the table from which the data was acquired, having the last known position automatically show its relating data through the marker, which is the time that it was taken on. The user is able to freely choose which markers should have their associated data displayed from there on out, by clicking on the desired ones. It is also important to note that the geographical data is truncated in the same way that the data seen in the graphs was, meaning that only the markers set in the specified time frame will be represented within the world map.

The map was zoomed in slightly so that the geographical progression

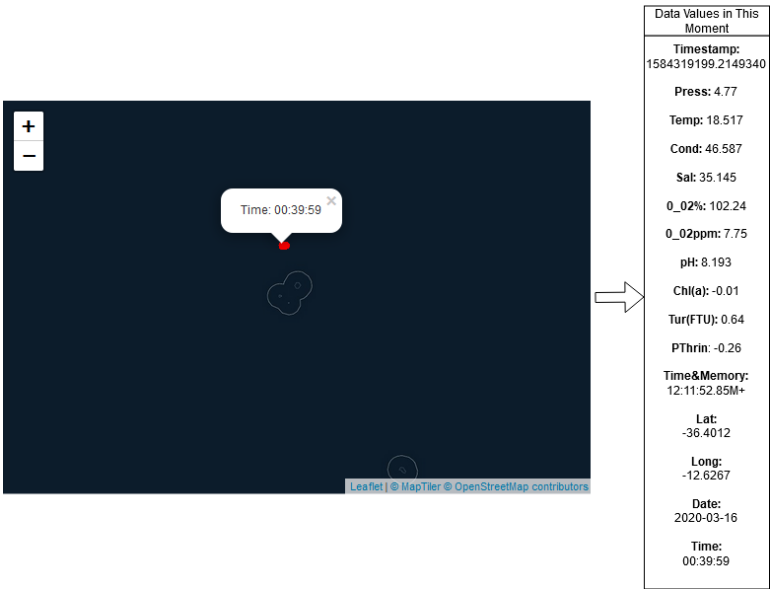


Figure 5.31: "TowFish" Map Data Visualization + Overall Current Data Values.

can be observed, given the sheer volume of geographical positioning data provided during the set period of time.

With that said, the data values shown in the added note in Figure 5.31 are in accordance to the data seen in the graphs and thus, the data line they were fetched from in the database, as can be seen in the highlighted data line in Figure 5.32.

5.3 Full Application Example

Lastly, a full data implementation example will be demonstrated. This application includes both the process of database editing, through the "Database Tool", and online visualization, through the "Online Data Viewer".

To begin with, the data used was the data taken from the "FeedFirst" autonomous system. This data was acquired over 4 days, generating one file of data output per hour over this time. A snippet of one of these log-files can be seen in Figure 5.33.

1584316822.062399	Press	Temp	Cond	Sal	O_O2%	O_O2ppm	pH	Chl(a)	Tur(...	PThrin	Time&Memory	Lat	Long	Date	Time
1 584 319 184,7825550	5	18,514	46,582	35,144	102,23	7,75	8,193	-0,14	0,55	-0,26	12:11:38.42M+	-36,4012	-12,6261	2020-03-16	00:39:44
1 584 319 185,8225870	5,05	18,511	46,58	35,144	102,18	7,75	8,193	-0,09	0,68	-0,12	12:11:39.37M+	-36,4012	-12,6261	2020-03-16	00:39:45
1 584 319 186,8466180	5,08	18,509	46,578	35,144	102,18	7,75	8,192	-0,08	0,62	-0,21	12:11:40.48M+	-36,4012	-12,6261	2020-03-16	00:39:46
1 584 319 187,8866630	5,02	18,507	46,578	35,146	102,2	7,75	8,192	-0,04	0,56	-0,2	12:11:41.43M+	-36,4012	-12,6262	2020-03-16	00:39:47
1 584 319 188,8946710	4,94	18,512	46,582	35,144	102,2	7,75	8,193	-0,08	0,7	-0,05	12:11:42.54M+	-36,4012	-12,6262	2020-03-16	00:39:48
1 584 319 189,9347110	4,97	18,515	46,586	35,146	102,19	7,75	8,192	-0,11	0,63	-0,16	12:11:43.49M+	-36,4012	-12,6263	2020-03-16	00:39:49
1 584 319 190,9587310	4,94	18,515	46,587	35,146	102,19	7,75	8,192	0,06	0,75	-0,27	12:11:44.60M+	-36,4012	-12,6263	2020-03-16	00:39:50
1 584 319 192,0147560	4,89	18,518	46,589	35,146	102,14	7,74	8,192	-0,1	0,68	-0,34	12:11:45.37M+	-36,4012	-12,6264	2020-03-16	00:39:52
1 584 319 193,0387890	4,84	18,519	46,588	35,144	102,14	7,74	8,192	0,14	0,54	-0,21	12:11:46.67M+	-36,4012	-12,6264	2020-03-16	00:39:53
1 584 319 194,0788100	4,85	18,519	46,588	35,144	102,09	7,74	8,193	-0,06	0,6	-0,4	12:11:47.62M+	-36,4012	-12,6264	2020-03-16	00:39:54
1 584 319 195,1349880	4,88	18,517	46,587	35,145	102,09	7,74	8,192	-0,01	0,59	-0,36	12:11:48.73M+	-36,4012	-12,6265	2020-03-16	00:39:55
1 584 319 196,1268840	4,86	18,517	46,587	35,145	102,16	7,75	8,193	-0,01	0,71	-0,4	12:11:49.68M+	-36,4012	-12,6266	2020-03-16	00:39:56
1 584 319 197,1508910	4,84	18,517	46,587	35,145	102,16	7,75	8,193	-0,01	0,66	-0,09	12:11:50.79M+	-36,4012	-12,6266	2020-03-16	00:39:57
1 584 319 198,1909170	4,78	18,516	46,586	35,145	102,24	7,75	8,193	0,06	0,61	-0,49	12:11:51.74M+	-36,4012	-12,6267	2020-03-16	00:39:58
1 584 319 199,2149340	4,77	18,517	46,587	35,145	102,24	7,75	8,193	-0,01	0,64	-0,26	12:11:52.85M+	-36,4012	-12,6267	2020-03-16	00:39:59
1 584 319 200,2549760	4,73	18,516	46,585	35,144	102,15	7,75	8,192	0,08	0,61	-0,36	12:11:53.80M+	-36,4012	-12,6268	2020-03-16	00:40:00
1 584 319 201,2630040	4,59	18,514	46,584	35,145	102,15	7,75	8,193	-0,11	0,55	-0,22	12:11:54.91M+	-36,4012	-12,6268	2020-03-16	00:40:01
1 584 319 202,3030440	4,48	18,514	46,585	35,145	102,13	7,74	8,192	0,08	0,69	-0,15	12:11:55.86M+	-36,4012	-12,6268	2020-03-16	00:40:02
1 584 319 203,3270450	4,41	18,516	46,585	35,144	102,13	7,74	8,192	0,3	0,61	-0,47	12:11:56.97M+	-36,4012	-12,6269	2020-03-16	00:40:03
1 584 319 204,3670860	4,4	18,516	46,586	35,145	102,18	7,75	8,192	0,1	0,6	-0,3	12:11:57.92M+	-36,4012	-12,6269	2020-03-16	00:40:04
1 584 319 205,3911270	4,2	18,514	46,585	35,146	102,18	7,75	8,192	-0,05	0,51	-0,22	12:11:59.03M+	-36,4012	-12,627	2020-03-16	00:40:05
1 584 319 206,4311680	4,01	18,512	46,582	35,145	102,25	7,75	8,193	-0,27	0,61	-0,18	12:11:59.98M+	-36,4012	-12,627	2020-03-16	00:40:06

Figure 5.32: Fetched Data Line (Last Line of Data Within Specified Time Frame). Visualized Through "HeidiSQL".

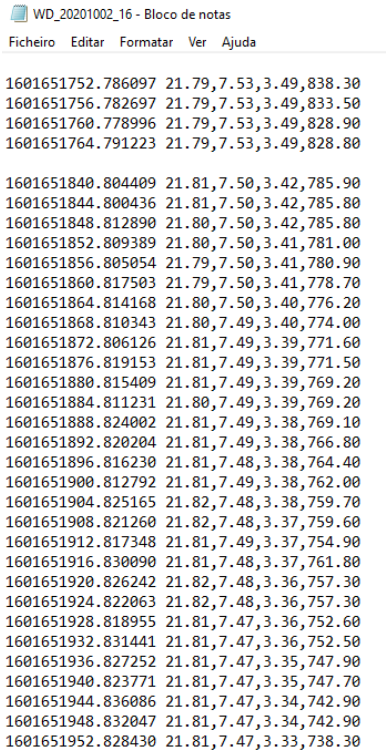


Figure 5.33: Full Application Log Snippet.

feedfirst_data.feedfirst: 28 537 rows total (approximately)

Timestamp	Temp	pH	Cond	Oxy	Date	Time
1 601 651 752,7860970	21,79	7,53	3,49	838,3	2020-10-02	16:15:52
1 601 651 756,7826970	21,79	7,53	3,49	833,5	2020-10-02	16:15:56
1 601 651 760,7789960	21,79	7,53	3,49	828,9	2020-10-02	16:16:00
1 601 651 764,7912230	21,79	7,53	3,49	828,8	2020-10-02	16:16:04
1 601 651 840,8044090	21,81	7,5	3,42	785,9	2020-10-02	16:17:20
1 601 651 844,8004360	21,81	7,5	3,42	785,8	2020-10-02	16:17:24
1 601 651 848,8128900	21,8	7,5	3,42	785,8	2020-10-02	16:17:28
1 601 651 852,8093890	21,8	7,5	3,41	781	2020-10-02	16:17:32
1 601 651 856,8050540	21,79	7,5	3,41	780,9	2020-10-02	16:17:36
1 601 651 860,8175030	21,79	7,5	3,41	778,7	2020-10-02	16:17:40
1 601 651 864,8141680	21,8	7,5	3,4	776,2	2020-10-02	16:17:44
1 601 651 868,8103430	21,8	7,49	3,4	774	2020-10-02	16:17:48
1 601 651 872,8061260	21,81	7,49	3,39	771,6	2020-10-02	16:17:52
1 601 651 876,8191530	21,81	7,49	3,39	771,5	2020-10-02	16:17:56
1 601 651 880,8154090	21,81	7,49	3,39	769,2	2020-10-02	16:18:00
1 601 651 884,8112310	21,8	7,49	3,39	769,2	2020-10-02	16:18:04
1 601 651 888,8240020	21,81	7,49	3,38	769,1	2020-10-02	16:18:08
1 601 651 892,8202040	21,81	7,49	3,38	766,8	2020-10-02	16:18:12
1 601 651 896,8162300	21,81	7,48	3,38	764,4	2020-10-02	16:18:16
1 601 651 900,8127920	21,81	7,49	3,38	762	2020-10-02	16:18:20
1 601 651 904,8251650	21,82	7,48	3,38	759,7	2020-10-02	16:18:24
1 601 651 908,8212600	21,82	7,48	3,37	759,6	2020-10-02	16:18:28

Figure 5.34: "FeedFirst" Data Table Snippet. Visualized Through "HeidiSQL".

As we can see, there were 5 different data columns, which relate to the time stamp, temperature, pH levels, conductivity and dissolved oxygen respectively, acquired during the usage of the "FeedFirst" autonomous system.

Through the "Database Tool", 2 of those days worth of "FeedFirst" log-files were inserted into a table in a fresh database. The generated table can be seen in Figure 5.34.

Given that there was a time stamp column in the log-files, it was used to generate the additional date/time columns seen in the figure through time stamp conversion.

With the data successfully stored, the next step was to setup the visual representation through the "Online Data Viewer". This includes setting up a "Dashboard" page and a "History" page that makes sense given the context.

The generated "Dashboard" page for the "FeedFirst" data visualization can be seen from Figure 5.35 to 5.37. In Figure 5.35 the gauges and text boxes used are represented, in Figure 5.36 the configured graphs are shown and lastly, in Figure 5.37 the dashboard's comparison graph is represented.

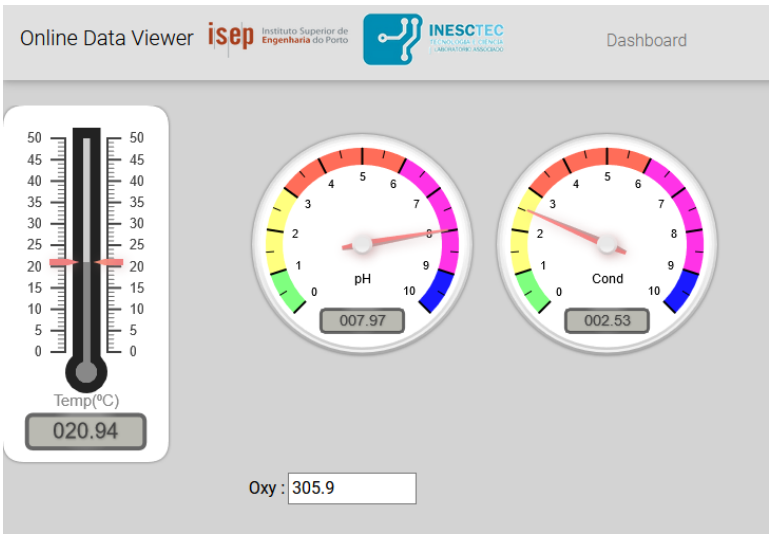


Figure 5.35: "FeedFirst" Dashboard Gauges.

All of the represented graphs, including the comparison graph, were configured to show 250 of the latest data entries in this case.

Finally the "History" page created for the context of the "FeedFirst" autonomous system can be seen in 5.38.

This page was created with data completely over a single table in mind. Resulting in an "History" page layout that connects immediately to the table that is to be used as soon as the page is opened. With the connection to the table established, the page asks its users if they want to specify the starting date/time and/or the ending date/time for data visualization, along with the number of graphs to be shown per row.

In order to assist the users in figuring out which dates and times have data, a small text box in the side of the page is generated which specifies for which dates and times for which data exists, given the table data acquired.

The data visualization for the History page was setup between 2 days, 2020-10-02 and 2020-10-03, starting at 16 : 30 : 15 of the first day up until 17 : 10 : 00 of the second day. As for the graphs per row, 2 of them were setup to be shown, as can be seen in Figure 5.39.

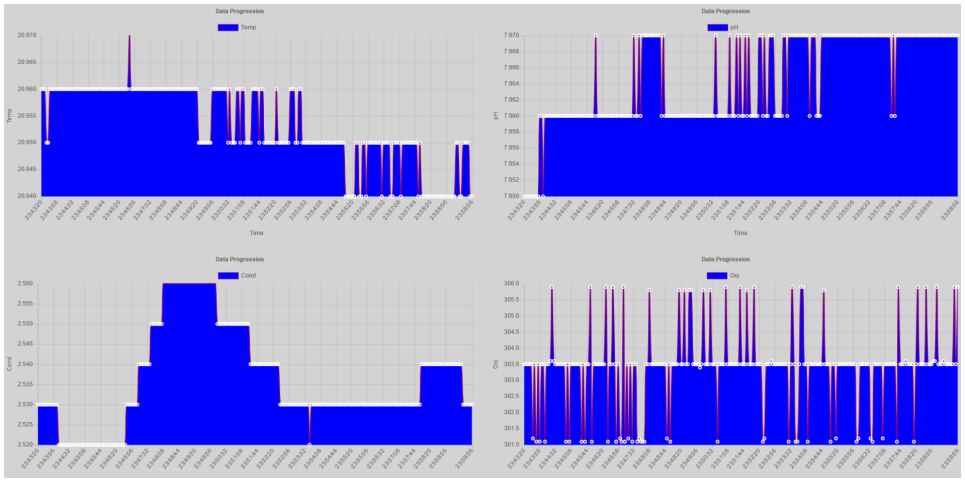


Figure 5.36: "FeedFirst" Dashboard Graphs.

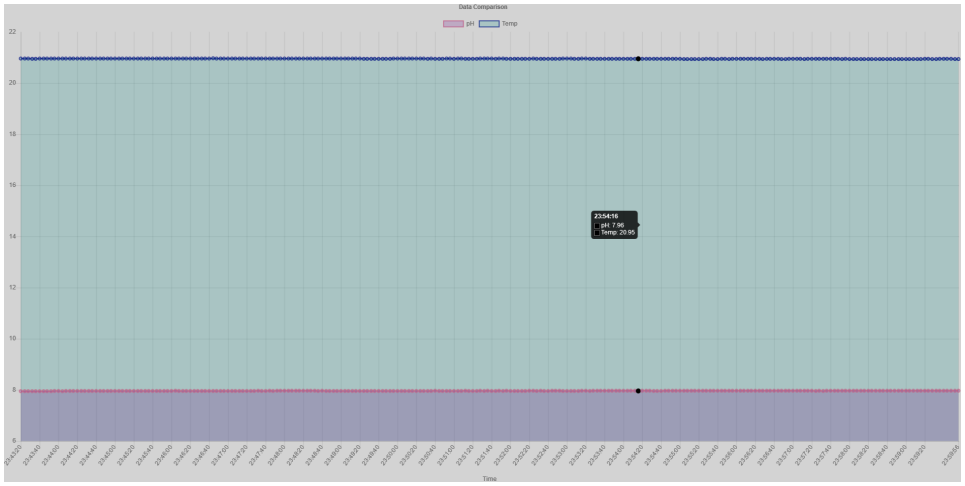


Figure 5.37: "FeedFirst" Dashboard Comparison Graph.

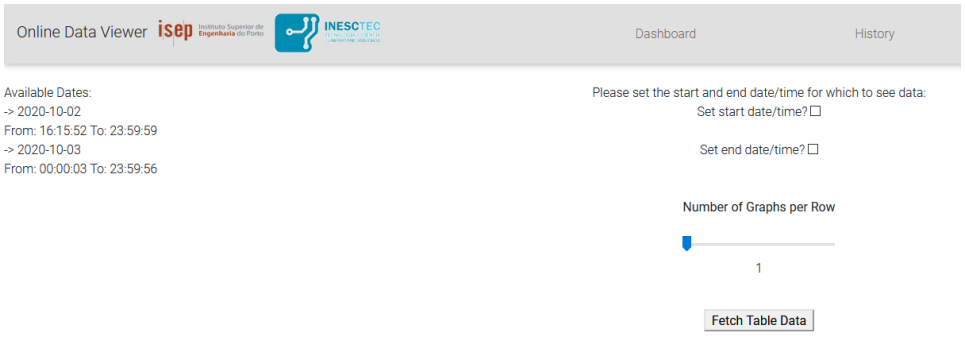


Figure 5.38: "FeedFirst" History Page.

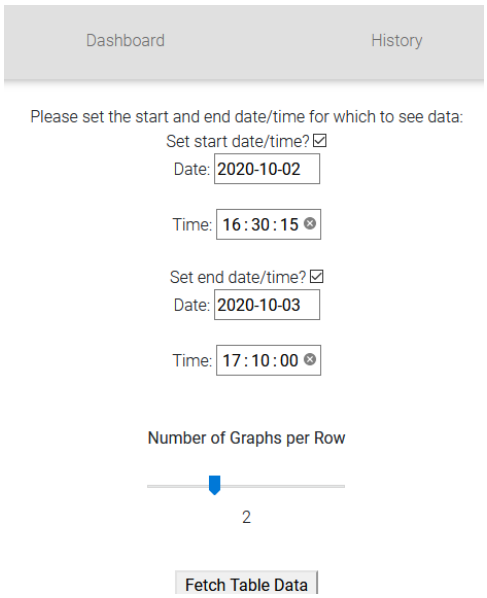


Figure 5.39: "FeedFirst" History Page Configuration.

When it comes to the data itself, the resulting graphs in this page can be seen in Figure 5.40, whilst the resulting comparison graph can be seen in Figure 5.41.

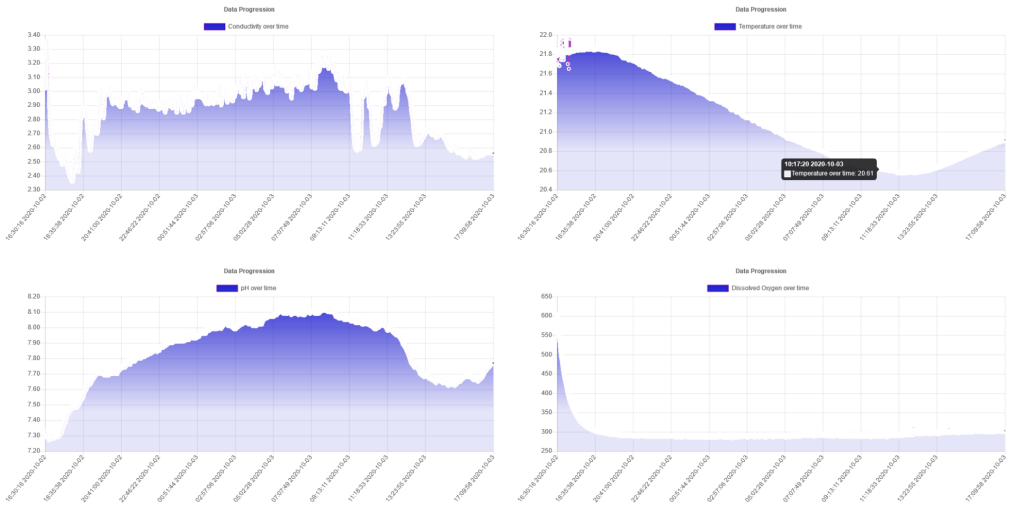


Figure 5.40: "FeedFirst" History Page Configured Graphs.

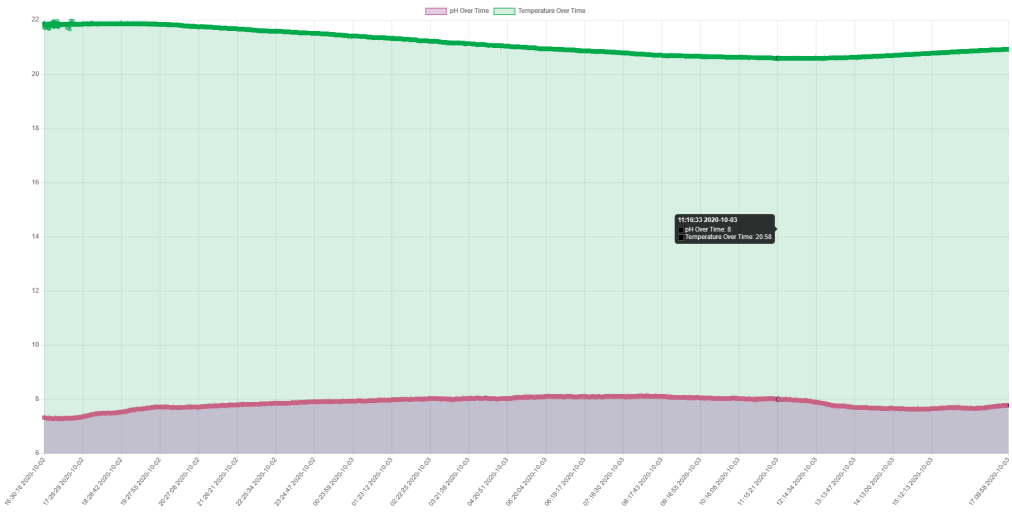


Figure 5.41: "FeedFirst" History Page Configured Comparison Graph.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

Concluding, the objectives set by this thesis were successfully fulfilled in a general sense.

The overall objective of the developed solution was to create a set of tools that could store and visualize the data outputs of a plethora of different sensors/groups of sensors, regardless of whether or not the application of those sensors was fully developed. As we can see, by analyzing the full application example provided in Chapter 5, both components of the solution were successfully implemented in an autonomous system that was still under development, namely the "FeedFirst" autonomous system, which facilitated testing of all the sensors that comprise it.

The necessary tool ("Database Tool") capable of both reading log-files with different structures and acquiring data in real-time directly from sensors, followed by writing the read data into tables in a database, was implemented with all the fundamental features listed.

On top of that, an online visualization tool ("Online Data Viewer") capable of parsing data stored in databases and showing it to an user in an easy to read fashion was successfully developed.

The solution concluded as a very robust and generally applicable data storage and visualization tool.

The customization options in the developed "Database Tool" allow for a great degree of freedom when it comes to parsing data output from any sensor or interface of sensors and its subsequent storage into tables inside a database.

On top of that, the developed "Online Data Viewer" not only can be applied to any and all numerical data stored in a database, as it also allows its users to freely customize the way they see data in real-time, for both themselves and other users of the same solution in a network.

The overall solution however, could benefit from better visual design decisions.

The "Database Tool" visually is fundamentally a series of console inputs, making it so an actual graphical interface to run the solution would make it not only more pleasing to look at, but also make it easier to use by someone who has no idea how it fundamentally works.

On the other hand, the "Online Data Viewer" does have a functional visual design, however said design could be improved through some additional "front-end" web development work and expertise.

As a final note and thanks to the groundwork established in the overall solution at this point, additional features, both visual and functional, can easily be added further down the line, being time the main limiting factor in that process.

6.2 Future Work

Even though the solution's development was considered successful and besides the possible visual design improvements mentioned, there can always be even more improvements made to it.

One of these possible improvements is the addition of the ability to read

and store images in databases out of the "Database Tool" solution. Through this, the "Online Data Viewer" could also be changed so that the images could be fetched and parsed into an image box in a "Dashboard" page layout or "History" page preset if the user wished to do so.

Another possible improvement could be the addition to a login system to the "Online Data Viewer". As it stands, when an user alters the dashboard's settings, that alteration occurs for all users. By incorporating a login system and associating different privileges to different users, the solution's visualization options could be controllable and observed by some of its users, whilst only observable to others.

Another way of approaching the login system would be that each user would have their preferred visualization options stored in a server, having said options associated with their accounts.

This page was intentionally left blank.

Bibliography

[1] MarinEye. "marineye - a new concept of ocean observation". http://marineye.ciimar.up.pt/. Accessed on 2020-07-06.

[2] Ninox. "ninox cloud". https://ninox.com/en/products. Accessed on 2020-06-15.

[3] TeamDesk. "online database software to empower your business management.". https://www.teamdesk.net/. Accessed on 2020-06-19.

[4] CASPIO. "the low-code platform that transforms your business.". https://www.caspio.com/platform-overview/. Accessed on 2020-06-20.

[5] Grafana. "heidisql". https://grafana.com/grafana/. Accessed on 2020-09-27.

[6] SQL4automation. "how to setup a grafana dashboard step by step". https://www.youtube.com/watch?v=4qpI4T6_bUw, 2020. Accessed on 2020-09-28.

[7] Webviz. "visualizing robotics data in the browser". https://webviz.io/. Accessed on 2020-08-20.

[8] European Environment Agency. "interactive data viewers". https://www.eea.europa.eu/data-and-maps/data/data-viewers#c0=20&c5=&b_start=0. Accessed on 2020-08-26.

[9] Center for Systems Science and Engineering. "covid-19 dashboard". https://gisanddata.maps.arcgis.com/apps/opsdashboard/index.html#/bda7594740fd40299423467b48e9ecf6. Accessed on 2020-08-27.

[10] Pedro Geraldes, Joel Barbosa, Alfredo Martins, André Dias, Catarina Magalhães, Sandra Ramos, and Eduardo Silva. In situ real-time zooplankton detection and classification. In *OCEANS 2019-Marseille*, pages 1–6. IEEE, 2019.

[11] INESC TEC. "sail project". https://twitter.com/sail_sagres/status/1272912879960891400. Accessed on 2020-09-09.

[12] Exame Informática. "volta ao mundo em eletricidade". https://www.inesctec.pt/download/1096/bb46afd6d5640e2e40d9feaf6f1cb621/01_01_2020_volta_ao_mundo_em_electricidade_exame_informatica.pdf&usg=AOvVaw234_lj02A1j11FDX-DtXHR. Accessed on 2020-08-01.

[13] BIP INESC TEC Magazine. "projeto sail: O inesc tec a bordo do navio sagres". http://bip.inesctec.pt/noticias/projeto-sail-o-inesc-tec-a-bordo-do-navio-sagres/. Accessed on 2020-08-01.

[14] INESC TEC. "commercial mysql – different editions". https://www.inesctec.pt/en/projects/sail2020#intro. Accessed on 2020-08-01.

[15] CIIMAR. "what is marineye?". http://marineye.ciimar.up.pt/projectscope/. Accessed on 2020-07-31.

[16] Instituto Superior de Engenharia do Porto. "isep researchers integrate scientific mission of the sagres training ship". https://www.isep.ipp.pt/New/ViewNew/6206. Accessed on 2020-08-01.

[17] INESC TEC. "feedfirst". https://www.inesctec.pt/en/projects/feedfirst#about. Accessed on 2020-08-02.

[18] FireBird. "the firebird licenses". https://firebirdsql.org/manual/qsg2-firebird-licenses.html. Accessed on 2020-07-31.

[19] MariaDB. "downloads". https://downloads.mariadb.org/. Accessed on 2020-08-14.

[20] MySQL. "commercial license for oems, isvs and vars". https://www.mysql.com/about/legal/licensing/oem/. Accessed on 2020-07-31.

[21] b lay. "commercial mysql – different editions". http://white-paper.b-lay.com/oracle-mysql-free-vs-commercial#!/java-commercial-features-are-required-to-be-licensed-but-when/. Accessed on 2020-07-31.

[22] SQLite. "sqlite is public domain". https://sqlite.org/copyright.html. Accessed on 2020-07-31.

[23] GNU. "gnu general public license, version 2". https://www.gnu.org/licenses/old-licenses/gpl-2.0.html. Accessed on 2020-07-31.

[24] Open Source Initiative. "the 3-clause bsd license". https://opensource.org/licenses/BSD-3-Clause. Accessed on 2020-07-31.

[25] Mozilla. "mozilla public license". https://www.mozilla.org/en-US/MPL/. Accessed on 2020-07-31.

[26] Jim Gray and Andreas Reuter. *Transaction processing: concepts and techniques*. Elsevier, 1992. Accessed on 2020-08-01.

[27] Methods Tools. "database locking: What it is, why it matters and what to do about it". https://www.methodsandtools.com/archive/archive.php?id=83. Accessed on 2020-08-01.

[28] Oracle. "using multiversion concurrency control". https://docs.oracle.com/database/bdb181/html/bdb-sql/mvcc.html. Accessed on 2020-08-02.

[29] IT Hare. "databases 101: Acid, mvcc vs locks, transaction isolation levels, and concurrency". http://ithare.com/databases-101-acid-mvcc-vs-locks-transaction-isolation-levels-and-concurrency/. Accessed on 2020-08-02.

[30] CUBRID. "transaction and lock". https://www.cubrid.org/manual/en/10.1/sql/transaction_index.html. Accessed on 2020-08-02.

[31] Database of Databases. "cubrid". https://dbdb.io/db/cubrid. Accessed on 2020-08-02.

[32] INGRES. "how mvcc works". https://docs.actian.com/ingres/11.0/index.html#page/DatabaseAdmin/How_MVCC_Works.htm. Accessed on 2020-08-02.

[33] INGRES. "ingres locking system". https://communities.actian.com/s/article/INGRES-Locking-System. Accessed on 2020-08-02.

[34] Hydromatic. "luciddbdatastorageandaccess". http://www.hydromatic.net/wiki/LucidDbDataStorageAndAccess. Accessed on 2020-08-02.

[35] Database of Databases. "luciddb". https://dbdb.io/db/luciddb. Accessed on 2020-08-02.

[36] MariaDB. "lock in share mode". https://mariadb.com/kb/en/lock-in-share-mode/. Accessed on 2020-08-02.

[37] MariaDB. "about xtradb". https://mariadb.com/kb/en/about-xtradb/. Accessed on 2020-08-02.

[38] SQLite. "sqlite for client-server". https://stackoverflow.com/questions/1321493/sqlite-for-client-server/1321593#1321593. Accessed on 2020-08-02.

[39] SQLite. "lsm users guide". https://sqlite.org/src4/doc/trunk/www/lsmusr.wiki#introduction_to_lsm. Accessed on 2020-08-02.

[40] SQLite. "write-ahead logging ". https://www.sqlite.org/wal.html. Accessed on 2020-08-02.

[41] Ninox. "frequently asked questions". https://ninox.com/en/manual/faq/faq. Accessed on 2020-06-14.

[42] Software Advice. "ninox software". https://www.softwareadvice.com/data-entry/ninox-profile/. Accessed on 2020-06-15.

[43] CASPIO. "build cloud apps without coding.". https://www.caspio.com/. Accessed on 2020-06-20.

[44] SolarWinds Msp. "data and server backup software". https://www.solarwindsmsp.com/products/backup. Accessed on 2020-06-22.

[45] Webviz. "lessons learned building a self-driving car on ros". https://roscon.ros.org/2018/presentations/ROSCon2018_LessonsLearnedSelfDriving.pdf. Accessed on 2020-08-20.

[46] European Environment Agency. "interactive data viewers". https://www.eea.europa.eu/about-us. Accessed on 2020-08-26.

[47] Esri. "arcgis online". https://www.esri.com/en-us/arcgis/products/arcgis-online/overview. Accessed on 2020-08-27.

[48] Guru99. "what is c programming language? basics, introduction and history". https://www.guru99.com/c-programming-language.html. Accessed on 2020-10-23.

[49] tutorialspoint. "database connectivity using c/c++". https://www.tutorialspoint.com/database-connectivity-using-c-cplusplus. Accessed on 2020-10-23.

[50] Pontifical Catholic University of Rio de Janeiro. "what is lua". https://www.lua.org/about.html. Accessed on 2020-10-23.

[51] tutorialspoint. "lua - database access". https://www.tutorialspoint.com/lua/lua_database_access.htm. Accessed on 2020-10-23.

[52] Python Software Foundation. "python is powerful... and fast; plays well with others; runs everywhere; is friendly easy to learn; is open.". https://www.python.org/about/. Accessed on 2020-10-23.

[53] Refsnes Data. "python mysql". https://www.w3schools.com/python/python_mysql_getstarted.asp. Accessed on 2020-10-23.

[54] Mozilla and individual contributors. "html: Hypertext markup language". https://developer.mozilla.org/en-US/docs/Web/HTML. Accessed on 2020-10-23.

[55] The PHP Group. "what is php?". https://www.php.net/manual/en/intro-whatis.php. Accessed on 2020-10-23.

[56] Refsnes Data. "what is css". https://www.w3schools.com/whatis/whatis_css.asp. Accessed on 2020-10-23.

[57] Mozilla and individual contributors. "about javascript". https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript. Accessed on 2020-10-23.

[58] Idronaut. "os311". https://www.idronaut.it/multiparameter-ctds/environmental-ctds/os311-environmental/. Accessed on 2020-10-27.

[59] Biral. "sws-050 visibility sensor". https://www.biral.com/product/sws-050-visibility-sensor/. Accessed on 2020-10-27.

[60] Alfredo Martins, André Dias, Eduardo Silva, Hugo Ferreira, Ireneu Dias, José Miguel Almeida, Luís Torgo, Marco Gonçalves, Maurício Guedes, Nuno Dias, et al. Marineye—a tool for marine monitoring. https://repositorio.inesctec.pt/bitstream/123456789/7148/1/P-00K-NAG.pdf, 2016. Accessed on 2020-07-06.

[61] MathWorks. "what is matlab?". https://www.mathworks.com/discovery/what-is-matlab.html. Accessed on 2020-10-24.

[62] Oxford University Press. "http daemon". https://www.oxfordreference.com/view/10.1093/oi/authority.20110803095948390?rskey=vgKutz&result=2. Accessed on 2020-09-11.

[63] Apache Software Foundation. "apache http server project". https://httpd.apache.org/. Accessed on 2020-09-11.

[64] Bootstrap Group. "build fast, responsive sites with bootstrap". https://getbootstrap.com/. Accessed on 2020-09-10.

[65] MDBootstrap.com. "material design for bootstrap". https://mdbootstrap.com/. Accessed on 2020-08-23.

[66] Canvas Gauges. "canvas gauges". https://canvas-gauges.com/. Accessed on 2020-08-23.

[67] Vladimir Agafonkin. "leaflet - an open-source javascript library for mobile-friendly interactive maps". https://leafletjs.com/. Accessed on 2020-08-21.

[68] OpenStreetMap. "openstreetmap". https://www.openstreetmap.org/. Accessed on 2020-08-21.

[69] MapTiler. "get started — maptiler cloud". https://cloud.maptiler.com/. Accessed on 2020-08-21.

[70] Ansgar Becker. "heidisql". https://www.heidisql.com/. Accessed on 2020-09-16.

Appendix A

Database Tool/Online Data Viewer Setup Guide

In this appendix chapter, the fundamental steps to install and utilize the 2 developed components of the solution, the "Database Tool" and "Online Data Viewer", will be mentioned, being that the steps presented in the case of the "Online Data Viewer" are provided in the context of installing the solution into a server, whilst explaining how to access it if a server is already running it. This includes installation steps for both supported operating systems: *Windows* and *Linux*.

A.1 Database Tool

A.1.1 Windows

Necessary installations:

- "MariaDB";
- *Python* Interpreter ("pycharm" used during development).

Once the initial installations have been concluded, please do the following.

1. Open the "main.py" file inside the project file with the *Python* interpreter of choice;

2. Make sure your *Python* interpreter's compiler is properly setup;

3. Alter the initial code relating to the installed "MariaDB" host, user and password (code snipped to be altered seen in Figure A.1), saving the changes afterwards;

4. Add the "mysql-connector" and the "mariadb" packages to the *Python* interpreter (add the "pip" and "setuptools" packages as well if they are not installed by default);

5. Run the solution, the changes made will be saved in the process.

Once these steps have been concluded, your *Python* interpreter is now ready to run the solution whenever you see fit.

Troubleshooting

→ **"mysql-connector" package does not seem to be working:**

Unfortunately, an extremely common issue. To solve this please add the "mysql-connector-python" and "mysql-connector-python-rf" packages to the *Python* interpreter of choice.

A.1.2 Linux

Necessary installations:

- "MariaDB";

- *Python* Interpreter ("python3" used during development).

Once the initial installations are concluded, please do the following:

1. Install the "mysql-connector" package ("*sudo pip3 install mysql-connector*");

```
#------ MariaDB Connection

mydb = mysql.connector.connect(
    host="localhost",
    user="root",
    passwd="maria123",
    database="ctd_data"
)
```

Figure A.1: "MariaDB" Connection Configuration Code.

2. Inside the "Database Tool" project directory, alter the initial code inside the "main.py" file relating to the installed "MariaDB" host, user and password (code snipped to be altered seen in Figure A.1), saving the changes afterwards;

3. Run the "main.py" file as a *Python* executable ("*sudo python3 main.py*").

Once these steps have been concluded, you should now be able to utilize the "Database Tool", repeating the third step each time you wish to do so.

Troubleshooting

→ **"mysql-connector" package does not seem to be working:**

Just like in the *Windows* operating system installation scenario, this is a very common issue. To solve this please try installing additional packages relating to the "mysql-connector" package, namely the "mysql-connector-python" and "mysql-connector-python-rf" packages. You can do so by running the following commands in the operating system's console:

- "*sudo pip3 install mysql-connector-python*";
- "*sudo pip3 install mysql-connector-python-rf*".

A.2 Online Data Viewer

A.2.1 Windows

Necessary installations:

- XAMPP;
- "MariaDB".

Once the previous 2 installations have been concluded, please do the following:

1. In the "XAMPP" installation folder, copy the "Online Data Viewer" project folder into the "htdocs" folder (the name of the "Online Data Viewer" folder can be changed at will);
2. Open the "XAMPP" control panel and start the "Apache" module;
3. Open the preferred browser and type "*localhost/[Solution Folder Name]*" into the address bar.

If these steps are concluded, any user inside the same network as the computer that is running the solution can now connect to the "Online Data Viewer", since the computer that is now running it is doing so as a server.

To access an "Online Data Viewer" running in a computer/server connected to the network, type the IPv4 address of the computer/server that is running the solution instead of the "localhost" in the address bar URL, resulting in an address along these lines: "*[Server IPv4 Address]/[Solution Folder Name]*".

A.2.2 Linux

Necessary installations:

- Apache;

- Apache PHP package;
- "MariaDB".

Once both the previous installations have been made execute the following steps:

1. Copy the "Online Data Viewer" folder to "*var/www/html*"(the name of the "Online Data Viewer" folder can be changed at will);
2. Open the preferred browser and type "*localhost/[Solution Folder Name]*" into the address bar.

If both steps have been properly taken care of, any user inside the same network as the computer that is running the solution can now connect to the Web Visualization Tool, since the computer that is now running it is doing so as a server.

To access an "Online Data Viewer" running in a computer/server connected to the network, type the IPv4 address of the computer/server that is running the solution instead of the "localhost" in the address bar URL, resulting in an address along these lines: "*[Server IPv4 Address]/[Solution Folder Name]*".

Troubleshooting

→ PHP is not being interpreted:

First and foremost, make sure the "Apache" PHP package is properly installed in your machine. If it still does not work, attempt the following:

1. Open the file located at "*etc/apache2/mods_available/php.7.4.conf*";
2. Look through the lines that block PHP parsing and comment them (these lines are explicit in the comments of the file itself).

→ **Other issues:**

If everything previously stated has been done and the solution is still not running properly, please make sure your LAMP interface's individual components are working properly. Please keep in mind that "MariaDB" is the relational database management system used when doing this, instead of the more commonly referred to "MySQL" relational database management system in LAMP interface installation guides.

Appendix B

Database Tool Flowcharts

In this appendix chapter, the flowcharts that describe some of the inner workings of the "Database Tool" will be shown.

The first 2 flowcharts, in Figures B.1 and B.2, summarize the overall functionality of the "Database Tool" through log-file parsing and real-time data acquisition respectively.

On the other hand, the flowchart seen in Figure B.3 is in regards to the process of date/time configuration, whilst the flowchart in Figure B.4 relates to the process of header/value line parsing during the writing of data into a table inside a database.

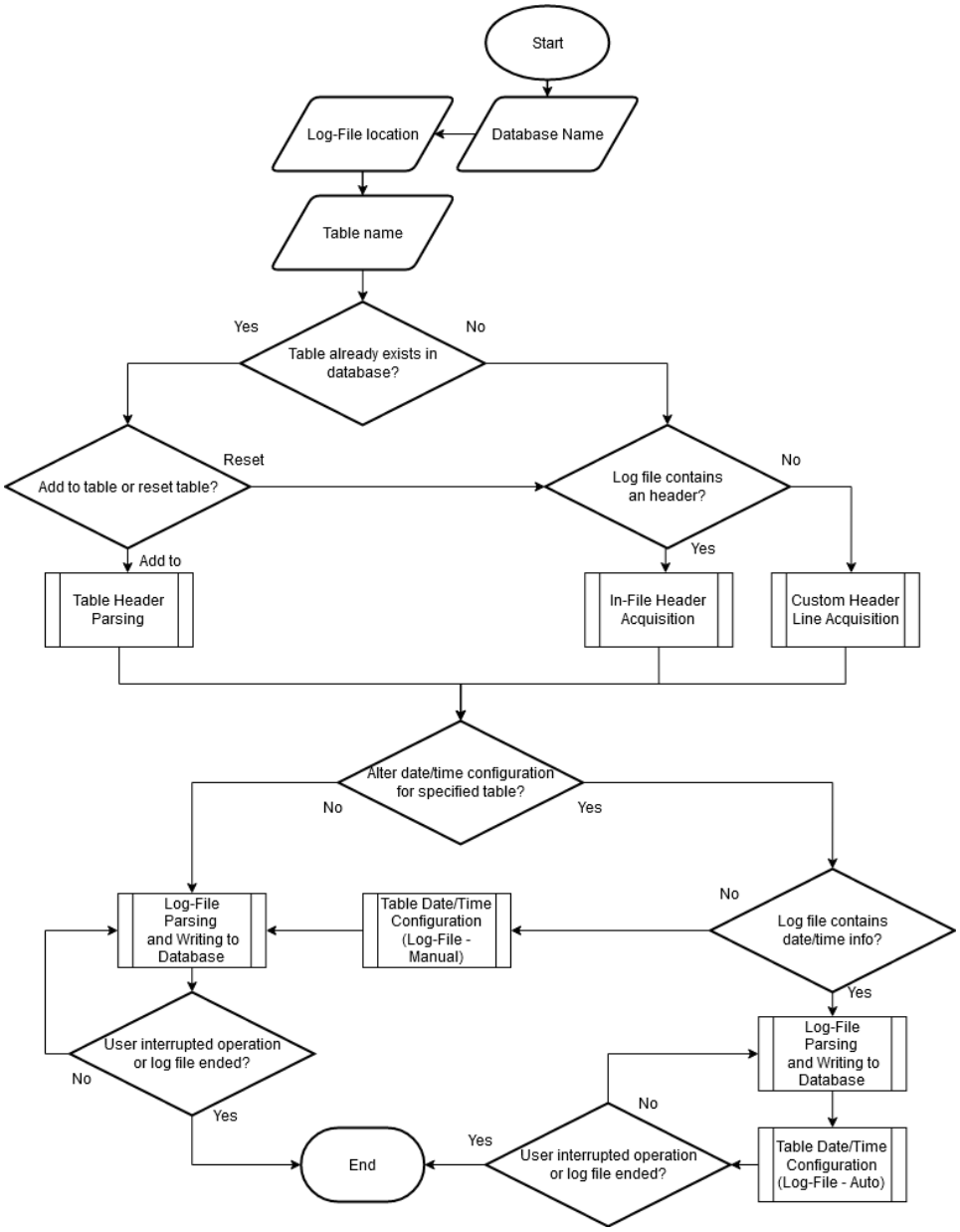


Figure B.1: Overall Database Tool Summary Flowchart (Log-File Parsing).

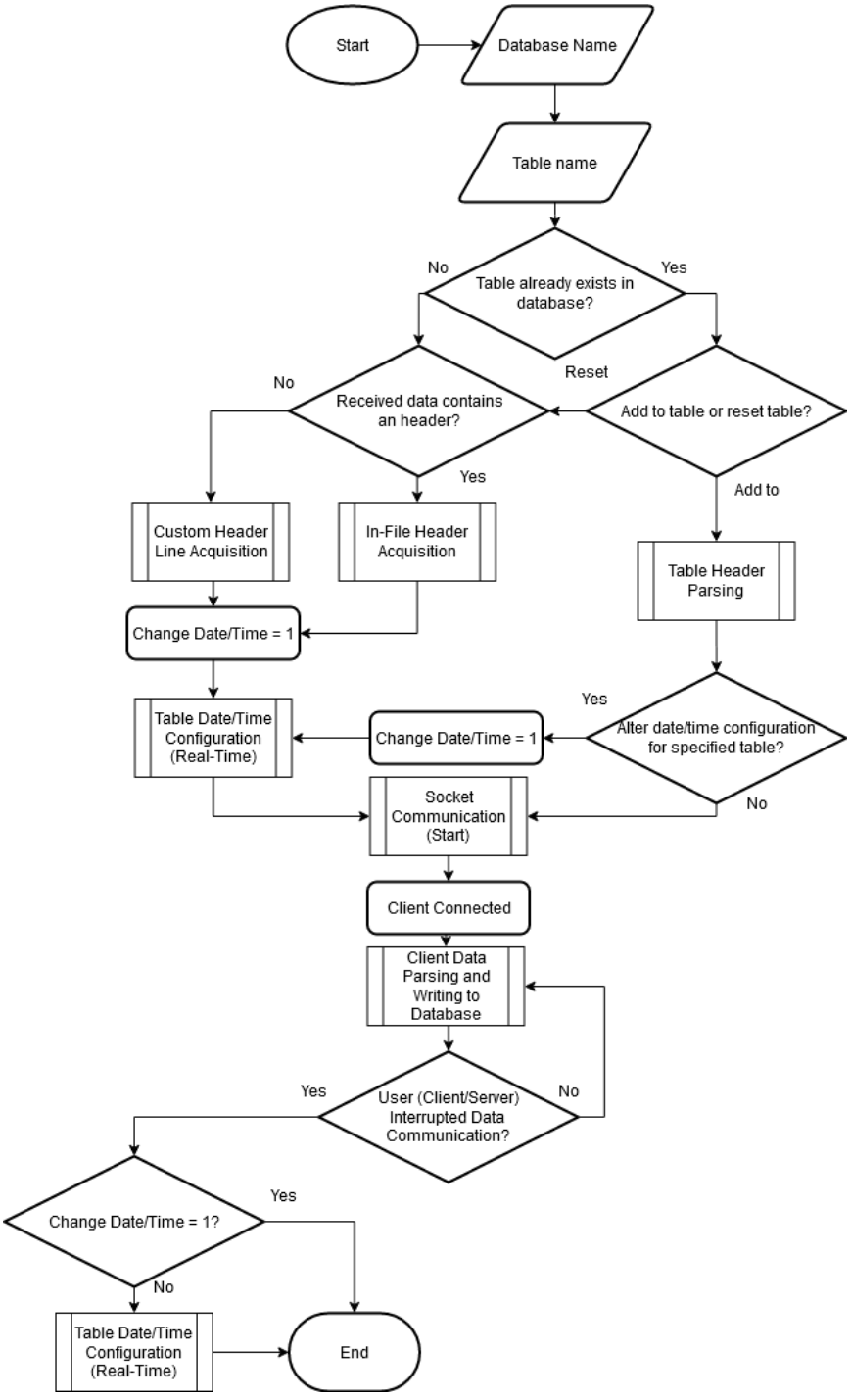


Figure B.2: Overall Database Tool Summary Flowchart (Real-Time Data Acquisition.)

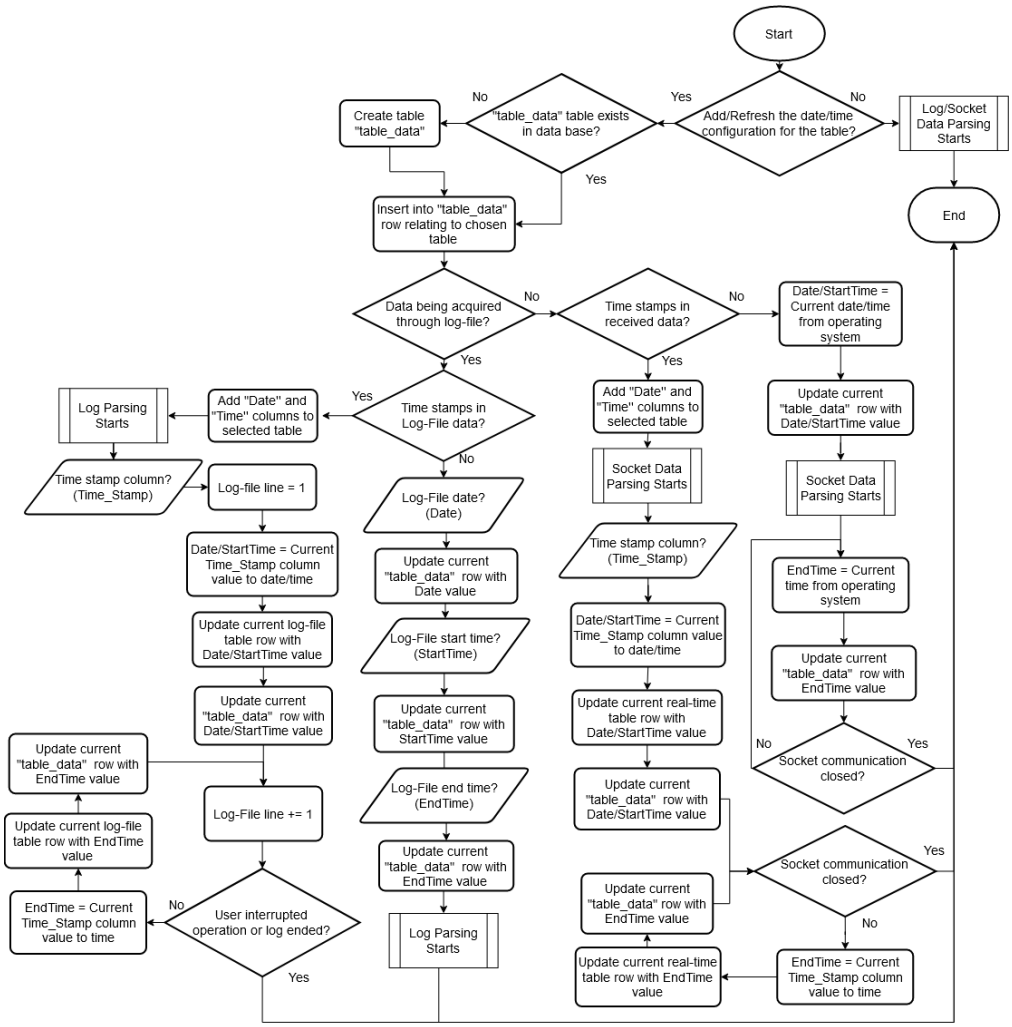


Figure B.3: Date/Time Data Acquisition Flowchart.

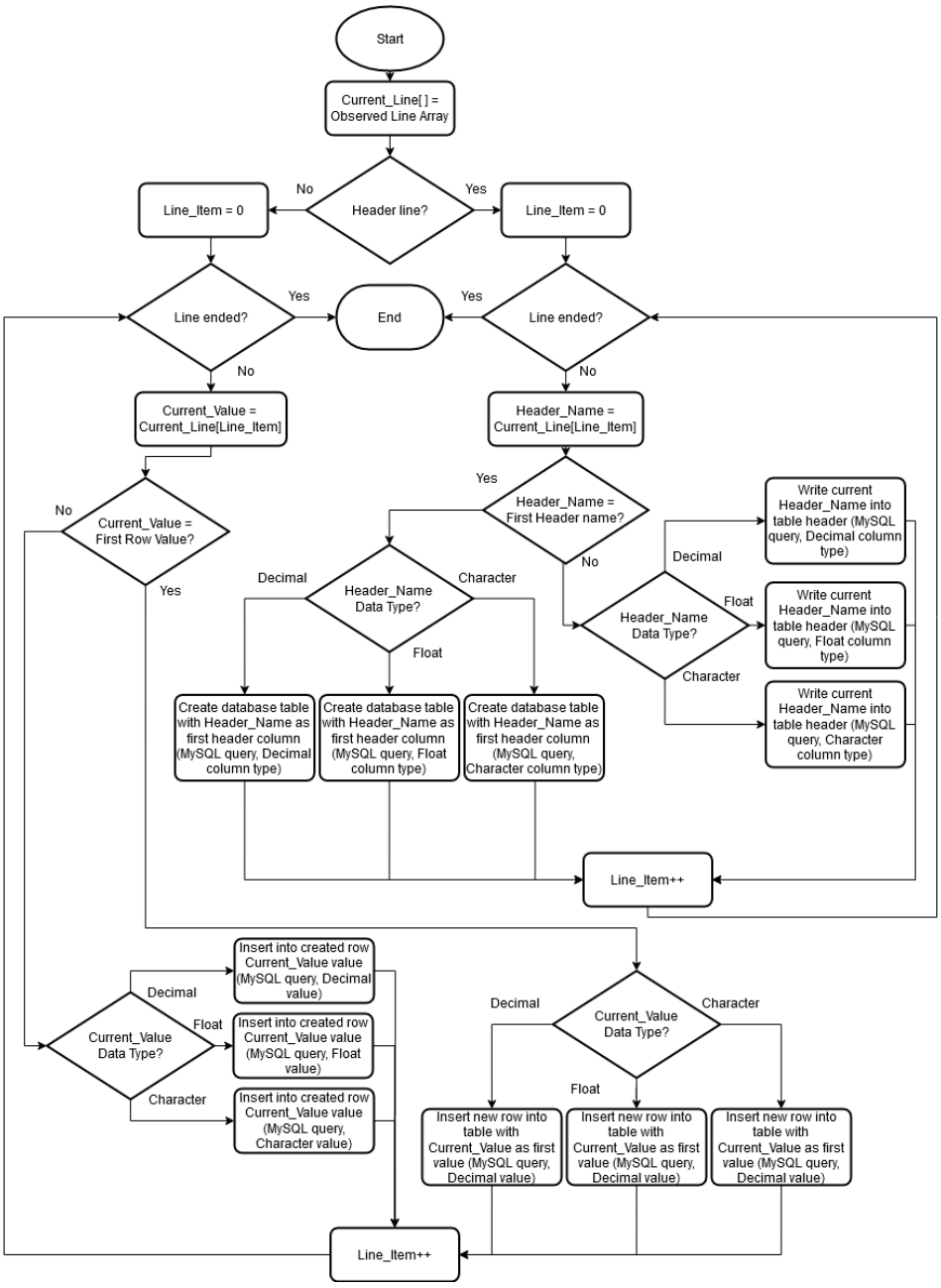


Figure B.4: Header/Value Line Parsing.