Universidad de Oviedo
Universidá d'Uviéu
University of Oviedo

ESCUELA POLITÉCNICA DE INGENIERÍA DE GIJÓN

GRADO EN INGENIERÍA EN TECNOLOGÍAS Y SERVICIOS DE TELECOMUNICACIÓN

Área de Ingeniería Eléctrica

TRABAJO FIN DE GRADO

Monitoring of a Photovoltaic Plant through a Web Portal

Pablo Fernández Pita

TUTOR: D. Manuel Arsenio Barbon Álvarez

TUTOR: D. Luis Froilán Bayón Arnau

FECHA: Julio de 2023

Contents

List of Figures

List of Tables

List of Code Fragments

Acronyms

AC Alternating Current. 6, 8, 21

API Application Programming Interface. 51, 57, 58, 60, 61, 66, 67, 68, 69, 70, 73, 74, 77, 78, 79, 80, 81, 82, 92

CRUD Create, Read, Update, Delete. 74

CSV Comma-Separated Values. 15, 18, 19, 25, 26, 27, 30, 31, 36, 38, 40, 43, 46, 47, 48, 54, 57, 58, 66, 67, 68, 87

DC Direct Current. 6, 8, 21

DOM Document Object Model. 62, 72

EPI Escuela Politécnica de Ingeniería de Gijón. 15

GUI Graphical User Interface. 71, 80, 82

HTML Hypertext Markup Language. 49, 62

HTTP Hypertext Transfer Protocol. 57, 66, 67, 68, 80

HTTPS Hypertext Transfer Protocol Secure. 81

IDE Integrated Development Environment. 42, 66, 78

IIS Internet Information Services. 79, 80, 81

JSON JavaScript Object Notation. 58, 61, 66, 67, 68

NaN Not a Number. 30, 31, 39

npm node package manager. 78

PV PhotoVoltaic. 1, 4, 6, 7, 12, 13, 22, 23, 42, 43, 46

REST Representational State Transfer. 61, 66, 67, 73, 77, 80, 81, 82

SSL Secure Socket Layer. 81, 82

THD Total Harmonic Distortion. 21

TLS Transport Layer Security. 81

UI User Interface. 49

URL Uniform Resource Locator. 48, 79

USB Universal Serial Bus. 18

UTC Coordinated Universal Time. 18, 19, 36, 37, 38, 55, 60

VPN Virtual Private Network. 53, 54

1. Introduction

In recent years, there has been a growing interest in renewable energy sources as a means of mitigating the effects of climate change and reducing reliance on fossil fuels. Among the most popular sources of renewable energy are solar PhotoVoltaic (PV) panels, which convert sunlight into electricity. As the cost of PV panels continues to decline, their adoption has become increasingly widespread, particularly in the residential and commercial sectors.

However, with this increased adoption comes the need for efficient monitoring and management of PV systems to ensure optimal performance and longevity. This is where the focus of this project lies - on the monitoring of PV panels. Through a series of sensors placed in the photovoltaic park, it is possible to keep track of the efficiency of the PV panels and identify possible problems that could arise. In order to interact with this data, one of the most popular solutions is to access an app or a website dedicated to showing the current state, as well as the evolution of the data, gathered.

Some of the more popular providers of these services are SolarEdge [1], Fronius [2] and SMA Solar [3]. These companies do not only sell high-efficiency solar panels and inverters but also the tools to monitor them easily. As seen in figures **1.1** and **1.2**, these services are mainly focused on delivering value to their customers. They are also keen on reassuring the decision they made when hiring their products by showing them clear metrics such as money saved on electricity or CO_2 not emitted to the atmosphere.

The monitoring in this project will be conducted for academic purposes, specifically focusing on the power generation process and the efficiency of the panels. By understanding the complexities of PV technology, one can gain a better appreciation for the benefits of this technology and contribute to the advancement of renewable energy source adoption.

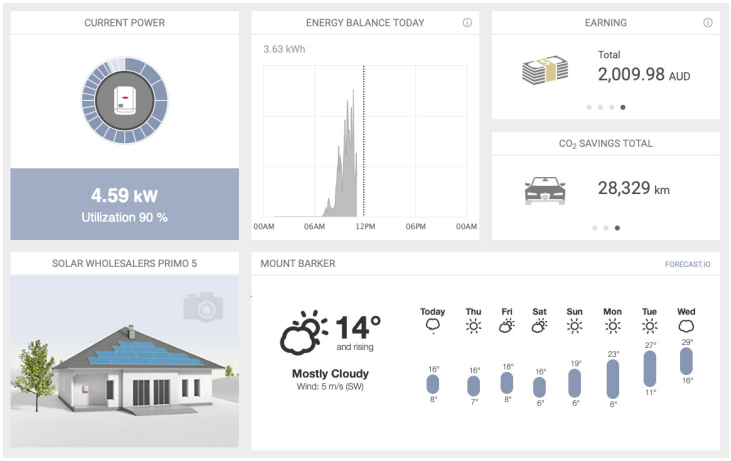


Figure 1.1.- Fronius monitoring service [4]

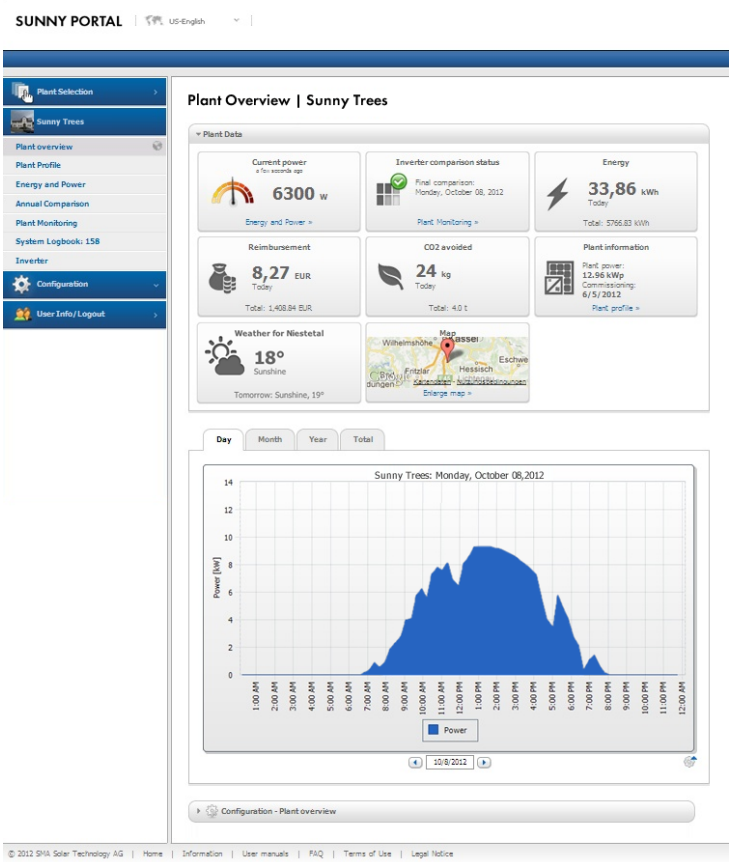


Figure 1.2.- Solar SMA monitoring service [5]

1.1.- Motivation for the project

As a final-year engineering student passionate about data science and renewable energies, I am really happy to contribute to the studies performed on the domain of solar energy production done at the campus by the university teachers.

This project serves as an ideal culmination of my academic journey, enabling me to apply the knowledge and skills acquired throughout my engineering degree as well as learn new useful skills by myself to fulfil a challenging task. By integrating the acquisition of data, its cleaning, storage, and subsequently visualization on a dedicated website, I will be able to showcase my proficiency in handling real-world engineering challenges.

Furthermore, by utilizing design patterns and good coding practices learned during my stay in college I also hope to lay solid foundations for future improvements on the system that will not cause problems scaling. The addition of new data sources, newer installations or more sensors on the existing ones is contemplated as it was encouraged by the professors leading me in this job.

1.2.- Project's goals

This final-year project aims to build a web service to monitor the state of the solar panels installed in the college, specifically the ones located behind the west department building. This project is part of a bigger investigation, currently expanding in Arganda del Rey, where more sensors are installed in another photovoltaic park. These two sites work together in an investigation performed by professors D. Luis Froilán Bayón Arnau and D. Manuel Arsenio Barbon Álvarez the instructors of this project.

The fundamental requirements that the website must abide by are:

- **Provide graphical and numerical information**: of the electrical energy generated and other relevant magnitudes
- **A system to access previously cleaned data**: in a format suitable to be analyzed with external mathematical tools such as Matlab or Mathematica. The

ability to select the dates and the variables present in this file is of upmost importance.

To provide this service, the PV monitoring system must have an architecture designed to fulfil these requirements as best as possible. In the figure 1.3 a schematic approach to this process is shown and the steps the data goes through before it can be shown on the website.

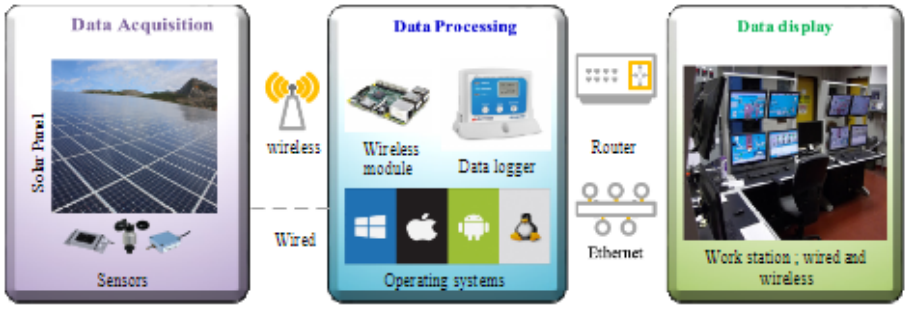


Figure 1.3.- A basic architecture of a solar PV monitoring system. Image obtained from [6]

1.3.- Document structure

Now that the goal and context of this project are set it is important to define how this document is going to be structured.

Firstly an overview of the technology and current state of solar energy generation will set a good base to understand the magnitudes treated and importance of the system. Next, in the third chapter, the process all the data follows before it can be accessed by the website is explained. Following that, the explanation of how the website works, the decisions made and the technologies used in its development. Finally, the deployment chapter will describe how everything was set up on the computer used as a server and how to access the application.

Lastly, a brief budget and planning guide of the project are shown followed by the conclusions and future improvements to be made to the system.

2. Photovoltaic plants

In Spain, a country blessed with an abundance of sunshine, solar plants play a pivotal role in the pursuit of clean and sustainable energy generation. The vast potential of harnessing solar power in this region has spurred significant investments in solar infrastructure. With a commitment to reducing carbon emissions and diversifying its energy sources, Spain has emerged as a leading player in the solar energy sector. This chapter focuses on the working principle of solar plants, the various categories of solar plants along with their constituent components.

2.1.- Working principle

The working principle of a solar cell is based on the phenomenon of the photovoltaic effect, which enables the conversion of sunlight directly into electricity. At the heart of a solar cell lies a semiconductor material, typically silicon, with specific chemical properties that facilitate the process. When sunlight, composed of photons, strikes the surface of the solar cell, the photons transfer their energy to the atoms in the semiconductor. This energy absorption promotes the generation of electron-hole pairs, where electrons are excited from their original positions, leaving behind positively charged holes. The built-in electric field within the solar cell then guides the separated electrons and holes in opposite directions. By placing metal contacts on the top and bottom surfaces of the cell, an external circuit can be completed, allowing the flow of electrons as an electric current. This flow of electrons represents the desired output of the solar cell, which can be used to power various devices or stored for later use. **Figure 2.1** shows a simplified diagram of this process.

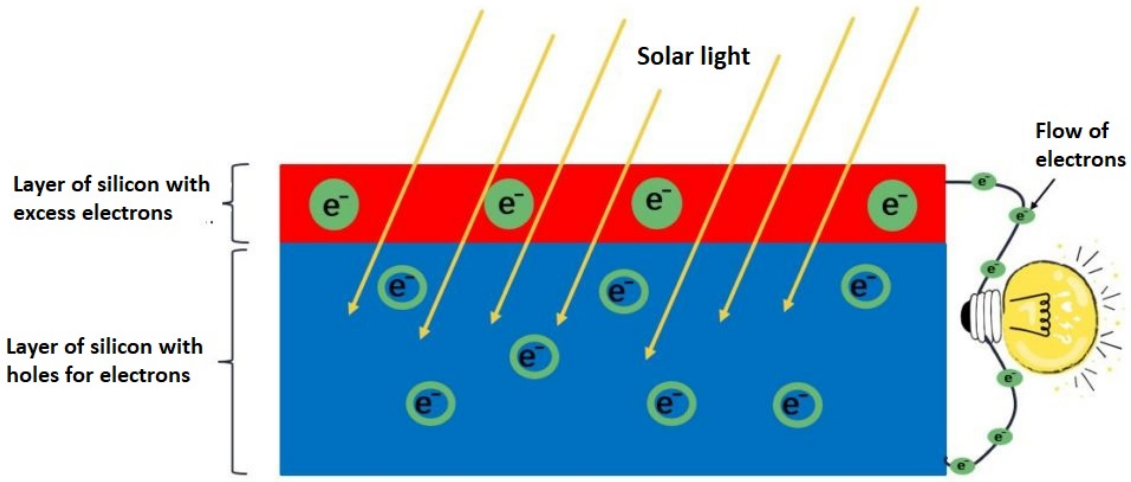


Figure 2.1.- Basic diagram of a solar cell. Image obtained from [7]

2.2.- Types of solar installations

Photovoltaic plants have emerged as crucial contributors to the renewable energy landscape, utilizing solar power to generate clean electricity. In this context, understanding the different types of PV plants is essential for optimizing their deployment and maximizing energy production. Two primary types of PV plants include utility-scale installations designed for large-scale electricity generation and distributed systems integrated into buildings for localized power generation. Each type offers unique advantages and contributes to the global transition towards a sustainable energy future.

2.2.1.- Utility-scale photovoltaic plants

Utility-scale PV plants are designed for large-scale electricity generation. These plants typically consist of a vast array of PV modules installed over a considerable land area. The modules are then connected in series and parallel to form strings and arrays, enabling higher voltage and power output. Utility-scale PV plants often employ tracking systems to maximize solar exposure throughout the day, enhancing energy production. These plants are integrated with power conditioning units, inverters, and transformers to convert the Direct Current (DC) power generated by the PV modules into Alternating Current (AC) power suitable for grid connection. Their extensive capacity and connection to the

grid make utility-scale PV plants essential contributors to renewable energy generation, supporting regional power demands and reducing reliance on fossil fuels.

2.2.2.- Distributed generation

On the other hand, there are also smaller-scale PV plants designed for distributed generation. These systems, often referred to as rooftop or building-integrated PV plants, are installed on residential, commercial, or industrial rooftops. Normally their orientation is fixed on a certain angle, calculated to maximize energy production throughout the year. They consist of a relatively smaller number of PV modules customized to fit the available roof space. Building-integrated PV plants offers several advantages, including the utilization of unused rooftop areas, reduced transmission losses, and proximity to energy consumers, minimizing grid infrastructure requirements.

These plants are typically grid-connected, allowing excess electricity to be fed back to the grid through net metering or utilized for local consumption. Distributed PV plants contribute to decentralizing the power generation landscape, promoting energy self-sufficiency, and reducing the carbon footprint at community level. Their modular design and adaptability make them suitable for various applications, ranging from individual residences to large commercial complexes.

2.2.3.- Off-grid installations

Another type of solar plant, usually found in remote locations and agricultural farms is off-grid installations. They operate independently and are placed to meet lighting demands, provide support for telecommunications, and power irrigation systems. **Figure 2.2** shows a diagram of an installation.

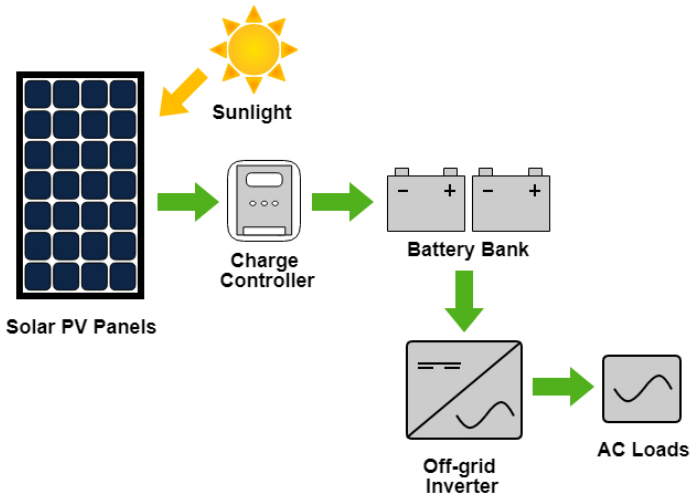


Figure 2.2.- Off-grid installation diagram. Image obtained from [8]

These installations rely on external batteries to have electricity during nighttime or on very cloudy days when no energy is been generated. If the energy generated from the plant is not enough to meet the demand, often a combustion engine is ready to replace the panels.

2.3.- Components of a solar plant

In order for a solar plant to work it needs to have the following components:

- **Solar panel**: the most visible element of our system. It consists of a group of photovoltaic cells in charge of capturing the solar radiation and converting it into a DC current.
- **Inverter**: converts the DC electrical current produced by the cells into AC current, available for consumption. It may come with an **optimizer** close to the panel itself to maximize the DC energy generated at each instant.
- **Transformer**: the alternating current generated by the inverters, is generally low voltage (300-800 V), for this reason, a transformer is used to elevate it to medium voltage (until 36kV) for transportation.

The solar plant will need two extra components to work properly in case of an off-grid set-up.

- **Battery**: to store the energy produced by the panels that is not consumed at the same time it is generated, the stored energy can then be used when needed, for example, at night.
- **Charge controller**: to protect the battery from overcharging and prevent inefficient use of the battery.

Figure 2.3.- PV modules in H configuration

2.3.1.- Current working setup at EPI Gijón

At this moment there are two strings installed on the plant, both connected to the grid. The two strings are connected to the inverter shown in **3.7**.

The first string is located next to the entrance of the solar park and it is composed of eight modules. It has sensors installed onto it that measure different types of irradiances, temperatures, and inclination. This string has one-axis tracking powered by motors from SunSlew [11]. The system is shown in **figure 2.4**.

Figure 2.4.- Tracker one's motor

The second string is now temporarily composed of eight panels equipped with one-axis tracking and two more from a "Mesa", in a south-oriented position and with a fixed inclination. This tracker also has the ability to shift horizontally over three rails to avoid shadows produced from the first string. This versatility also enables a wider variety of studies focused on shadows and reflections caused by different alignments between strings.

The position of each cell inside the string can be seen in **figure 2.5**. As the diagram shows, some panels are positioned horizontally and others vertically. As the analysis performed can be done module by module, this also allows greater flexibility in the investigations made with this string.

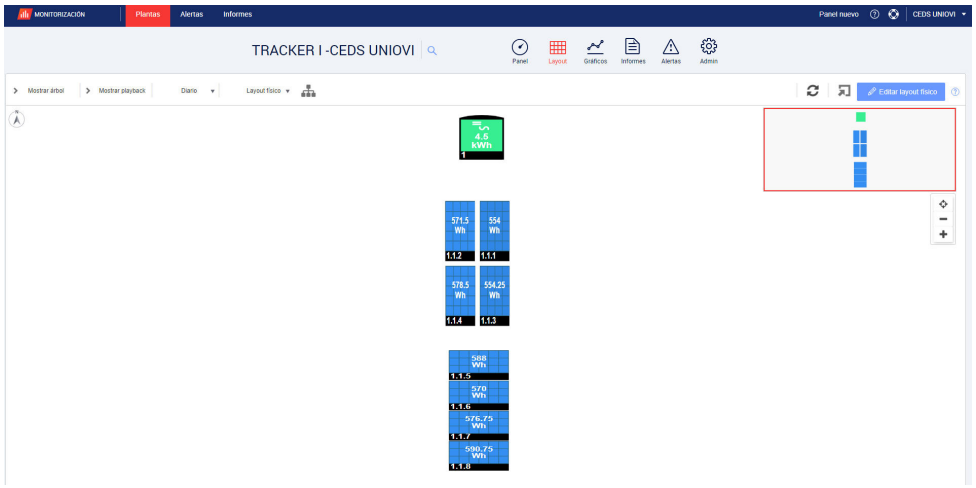


Figure 2.5.- Physical layout of the first tracker

The logical topology of this first string is shown in **image 2.6**. This image, obtained from the website of the inverter, shows all cells connected in series to the inverter. A typical string can only generate as much power as the worst panel. This poses a big problem in shaded environments as the total energy generated by our string will be much lower than it needs to be. However, with the current design of the system, with one power optimizer per module, if one panel is not producing as much power as it should, the rest of the system will adjust, by modifying the IV curve of the rest of the panels on the string to keep the input voltage at the inverter constant. With this solution, the loss in energy is mitigated and the system is more robust and tolerant to failures.

Figure 2.6.- Logical layout of the first tracker

The solar panels used for the plant are manufactured by CanandianSolar and their specifications are listed below on **table 2.1**

Table 2.1.- Specifications of Solar Cells

Variable	Value
Model	BiKu CS3U-350-365PB-FG
Nominal Maximum power	355 W
Optimum Operating Voltage	39.4 V
Optimum Operating Current	9,02 A
Open Circuit Voltage	46,8 V
Short Circuit Current	9.59 A
Efficiency	17,79%
Maximum Series Fuse Rating	20 A
Warranty	10 years

2.4.- Relevant magnitudes to monitor

When monitoring a solar plant, several variables play a crucial role in assessing its performance and ensuring optimal operation. The first essential variable is solar irradiance, which represents the intensity of sunlight reaching the photovoltaic (PV) panels. By continuously measuring solar irradiance, operators can evaluate the available solar resource and identify any deviations from expected levels. This information is vital for determining the potential energy production and diagnosing any issues that may affect the system's efficiency.

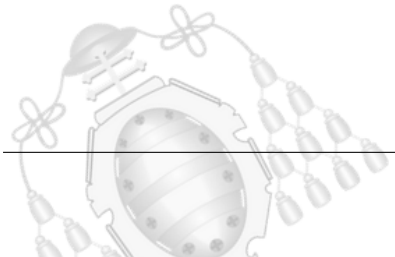


Figure 2.7.- Pyranometer used at our plant to measure irradiance

Another critical variable is the temperature of the PV panels. Solar panels are sensitive to temperature changes, and excessive heat can reduce their performance. Monitoring panel temperature allows operators to assess the impact of temperature on the electrical output and efficiency of the system. By tracking panel temperature in real-time, it becomes possible to implement cooling measures or adjust operating parameters to mitigate the negative effects of elevated temperatures.

Additionally, the electrical output of the solar plant is a fundamental variable to monitor. This includes measuring the current and voltage generated by the PV modules, as well as the power output of the entire system. By monitoring electrical parameters, operators can evaluate the overall performance of the solar plant, detect any abnormalities or deviations from expected values, and promptly address any issues that may arise.

Furthermore, monitoring the environmental conditions surrounding the solar plant is essential. Factors such as ambient temperature, humidity, and wind speed can impact the performance and efficiency of the system. Understanding the relationship between these environmental variables and the solar plant's operation allows for improved system management, such as optimizing energy generation during favourable weather conditions or implementing protective measures during extreme weather events.

As demonstrated by Hussein A Kazem and Miqdam T Chaichan, in their paper "Effect of Humidity on Photovoltaic Performance Based on Experimental Study" [9] higher humidity decreases solar panels' efficiency. The conclusion of this study states that relative humidity has an inverse strong correlation with current, voltage and power. In the case of power, the correlation factor R was -0.98395.

Figure 2.8 shows the results of their experimental investigation. This decrease in power due to higher humidity could be attributed to small water droplets, as well as water vapour, with the potential to accumulate on the surface of solar panels. Consequently, they can deflect or alter the path of sunlight, diminishing its direct impact on the solar cells. As a result, the reduced sunlight exposure leads to a decrease in electricity generation. This would mean that it can also be wise to measure the dew point, which represents the temperature at which air becomes saturated and condensation occurs, as part of the monitoring process.

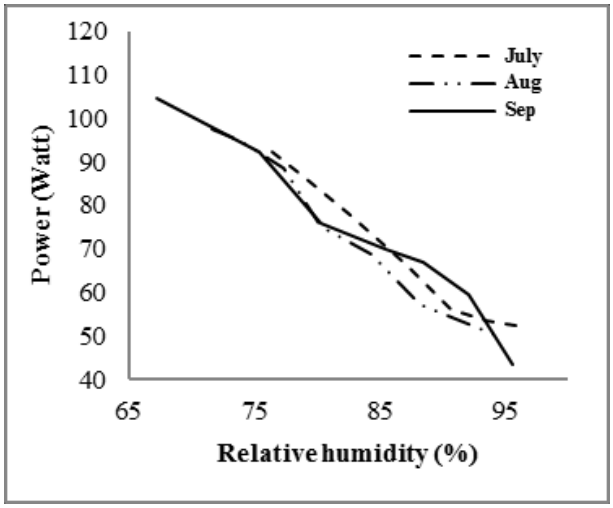


Figure 2.8.- Relationship between relative humidity and power output. Image obtained from [9]

In conclusion, the most important magnitudes to monitor in a solar plant include solar irradiance, panel temperature, electrical output, and environmental conditions. By continuously assessing these variables, operators can gain valuable insights into the system's performance, identify potential issues, and take proactive measures to optimize energy production and ensure the long-term reliability of the solar plant.

3. Data lifecycle

In order to obtain valuable insights from the data, it must first go through a process where the data is properly extracted, revised, and stored. This process was performed with two scripts written in Python, making use of two of its most commonly used libraries in the field of data science, Pandas and NumPy. One script for the data coming from the dataloggers and another for the records generated by the inverter's software.

3.1.- Data acquisition

The first stage in any data life-cycle is to obtain the data itself. This process is called data acquisition and it essentially encompasses the gathering of information to comprehend an electrical or physical phenomenon by utilizing sensors, measurement devices, and an electronic device capable of registering that information.

In this study's case, two dataloggers are available on-site that perform that last operation. One dedicated to the variables measured on the terrace, and the other to those located on the garden. Both dataloggers are located rather closely, as they are both inside the fourth module of the West department building of the Escuela Politécnica de Ingeniería de Gijón (EPI).

Both dataloggers are from manufacturer Kipp&Zonnen too, more specifically model LogboxSE [10], shown in **figure 3.6**. **Table 3.1** shows the characteristics of these devices. The sampling rate is set to one minute, and, as it is configured to create one Comma-Separated Values (CSV) file per day, that means it should produce 1440 values per day. Nevertheless, as it is a non-ideal world, where errors occur on a daily basis, the system must be ready to deal with corrupt, incomplete, or incorrect data.

The sensors are connected to both dataloggers by cable, but not all sensors use the same connections and have the same capabilities. Datalogger 1, also called "Estaci" is connected to a weather station manufactured by Lufft (WS700-UMB Smart Weather

Sensor) by a serial input and likewise to a Kipp&Zonnen RT1 (Smart Rooftop Monitoring System). A picture of this configuration can be seen in **figure 3.1**. It is also connected to two pyranometers by its analogue inputs. This datalogger is located on the rooftop of the West Department Building module 4 and provides many interesting magnitudes, all of which can be seen in **table 3.3**.

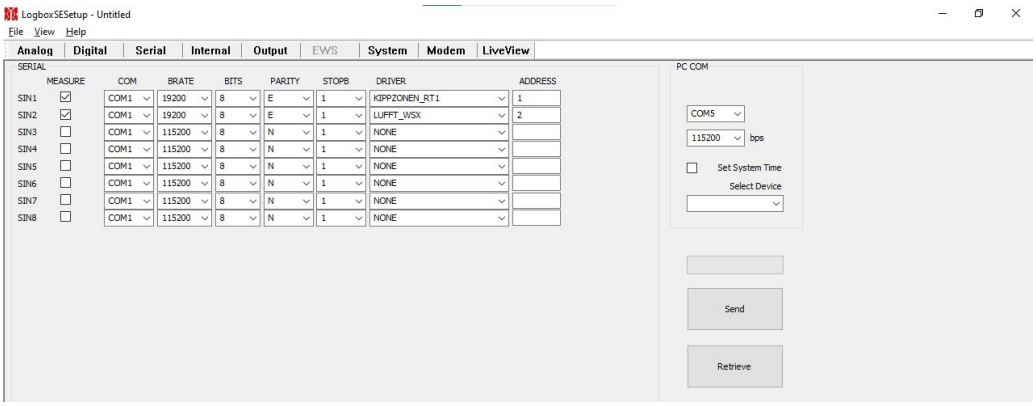


Figure 3.1.- Serial input datalogger 1

Figure 3.2.- Weather station WS700 by Lufft

Datalogger 2 on the other hand is located on the ground floor of the same building, behind the solar park. In this case, the connection between the sensors and the datalogger is done by underground cables that go into the "Sala de control" through an opening on the wall closest to the park.

Figure 3.3.- Underground installation for connection with control room

This second datalogger has a different function from the first one, as it gathers more information about the irradiation and temperatures on the actual site, as well as particles on the air, with each sensor having a different placement to compare set-ups and techniques. The variables currently measured can be seen in **table 3.4**.

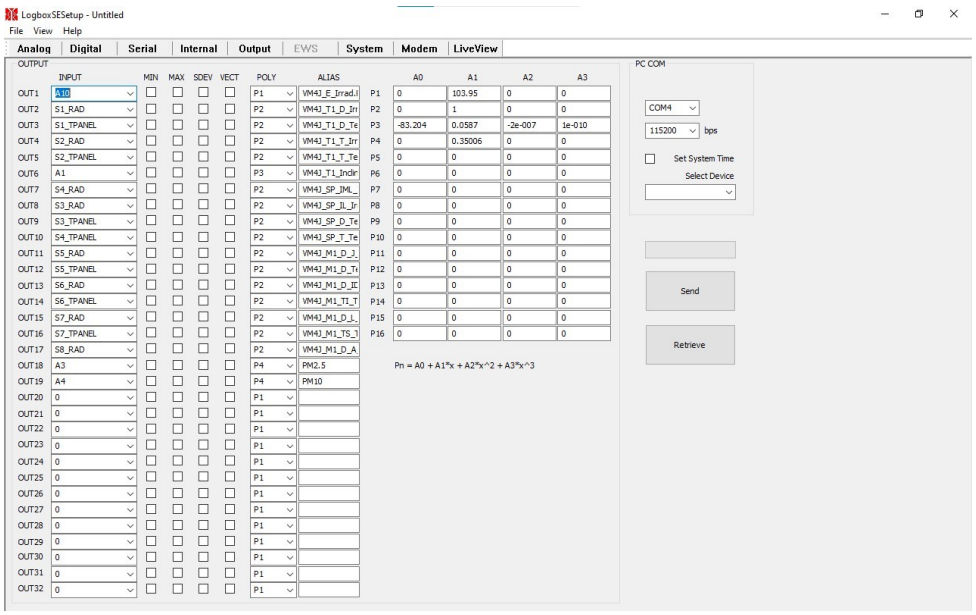


Figure 3.4.- Configuration of Datalogger 2. Output variables

In order to extract the data generated from these dataloggers they are connected via Universal Serial Bus (USB) to a computer, so you can extract the CSV files generated each day from the SD card of the datalogger. The setup is shown in **image 3.5**. This process is currently done manually, however, with a newer datalogger it could be performed automatically. Some important considerations regarding the extraction, if the datalogger is accessed when it is writing onto a file, it may corrupt the file or at least several rows. To prevent this, the dataloggers have a green light indicating the action it is performing.

Figure 3.5.- DC power source and computer connected to datalogger 2

This process limits the automation possibilities of the system. However as it is the technology available right now, it must get dealt with. In newer dataloggers, the files could be directly sent to a different computer without the need for human intervention but for the time being it will be kept as it was.

An important aspect when dealing with time-referenced logs is knowing which convention it uses regarding time zones and daylight saving time. In the case of this study, both dataloggers work with Coordinated Universal Time (UTC) times. This practice is widely adopted when working with solar measurements due to its consistency throughout

the year. Using UTC simplifies the handling of dates affected by daylight saving time adjustments, ensuring a more straightforward and reliable analysis of the data.

On the other hand, the software used by the inverter (SolarEdge [1]) only provides the timestamp related to the local time of the installation. This poses a problem, as one of the key elements of this study is the comparison of the irradiances provided by the sensors through the datalogger and the energy generated measured by the string inverter. In **section 3.3.1.1** the process of synchronizing these two sources will be explained in detail.

To download the raw data, the software provided by SolarEdge is far more modern and customizable than the one on the dataloggers and allows you to select the specific dates and magnitudes to download to a CSV. To access this data, you only need to connect to the application of SolarEdge with the credentials required and select the desired data. The inverter, however, only creates rows for hours when there was energy being produced, which will need to be sorted out by the system for its final storage and usage.

This behaviour is going to be similar to that of this monitoring system, but including the rest of the sources and with a standardised format for all.

3.1.1.- Specifications of the components involved

Figure 3.6.- LOGBOX SE DataLogger

Table 3.1.- Specifications on LogboxSE

Variable	Description
Analogue inputs	4 differential 24-bit and 4 single-ended 12-bit
Input ranges	19 mV to 2.5 V differential, 2 x 2.5 V and 2 x 3 V single-ended
Accuracy	0.05 % for 24-bit, 0.1 % for 12-bit
Digital inputs	4, maximum input 15 V, logic levels 3 x 3 V and 1 x 0.5 V
Serial input	RS-485 port for up to 8x compatible Modbus® devices from Kipp&Zonen, Lufft or IMT
Supply voltage DC	6 x AA internal batteries or external supply 4 to 24 V
Power consumption	1 mA standby, 7 mA typical during measurement, 50 mA with modem on
Memory card type	SD card (512 MB included)
Communication	RS-232, USB, quad-band GSM / GPRS modem with external antenna

Figure 3.7.- Inverter SE400H

Table 3.2.- Specifications of Inveter Wave SolarEdge Home SE400H

Variable	Description
Nominal Output Power AC	4000 VA
Nominal Output Voltage	220/230 V
Maximum output Current	18.5 A
Total Harmonic Distortion (THD)	less than 3%
Maximum Input Power	8000 W
Maximum Input Voltage DC	480 V
Maximum Input Current DC	11.5 A
Maximum Efficiency	99.2%
Weight	9 Kg
Dimensions	280 x 370 x 142 mm
Operating temperature range	From -40ºC to 60ºC

3.1.2.- Measured variables

As explained in **section 2.4** there are many variables that can be useful to monitor a solar plant's performance. At the moment, there are 13 and 19 variables being monitored on a daily basis on each datalogger. These variables are described in **tables 3.3** and **3.4**.

Table 3.3.- Variables measured on Datalogger 1

Parameter	Variable name	Units
Global irradiance over horizontal surface	VM4T_E_Irrad.Global_Avg	W/m^2
Diffuse irradiance over horizontal surface	VM4T_E_Irrad.Difusa_Avg	W/m^2
Ambient temperature	VM4T_E_T.Amb_Avg	ºC
Relative humidity	VM4T_E_Hum.Rel_Avg	%
Dew point	VM4T_E_P.Rocio_Avg	ºC
Wind speed	VM4T_E_Vel.Viento_Avg	m/s
Wind direction	VM4T_E_Dir.Viento_Avg	º dir. North
Atmospheric pressure	VM4T_E_P.atm_Avg	hPa/mBar
Precipitation	VM4T_E_Precip.int_Avg	mm
Precipitation per hour	VM4T_E_Precip_Avg	mm

Table 3.4.- Variables measured on Datalogger 2

Parameter	Variable name	Units
Reflected irradiance over horizontal surface	VM4J_E_Irrad.Reflejada_Avg	W/m^2
Total irradiance over Tracker 1, frontal part	VM4J_T1_D_Irrad.RT1_Avg	W/m^2
Total irradiance over Tracker 1, back part	VM4J_T1_T_Irrad.RT1_Avg	W/m^2
Temperature Tracker 1, frontal part	VM4J_T1_D_Temp.RT1_Avg	ºC
Temperature Tracker 1, back part	VM4J_T1_T_Temp.RT1_Avg	ºC
Angle of inclination of PV module over tracker	VM4J_T1_Inclinometro_Avg	º
Total irradiance of polar system at half latitude inclination	VM4J_SP_IML_Irrad.RT1_Avg	W/m^2

Continued on next page

Table 3.4 – continued from previous page

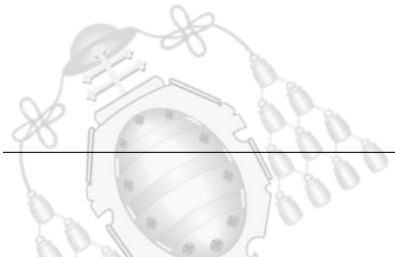
Parameter	Variable name	Units
Total irradiance of polar system at latitude inclination	VM4J_SP_IL_Irrad.RT1_Avg	W/m^2
Temperature of polar system, frontal part	VM4J_SP_D_Temp.RT1_Avg	ºC
Temperature of solar system, back part	VM4J_SP_T_Temp.RT1_Avg	ºC
Temperature of table 1, over frontal part of the PV module	VM4J_M1_D_Temp.RT1_Avg	ºC
Temperature of table 1, over frontal part of the PV module bottom	VM4J_M1_TI_Temp.RT1_Avg	ºC
Temperature of table 1, over frontal part of the PV module top	VM4J_M1_TS_Temp.RT1_Avg	ºC
Total irradiance of table 1, frontal part of the module, with optimal yearly angle proposed by IDAE	VM4J_M1_D_IDAE_Irrad.RT1_Avg	W/m^2
Total irradiance of table 1, frontal part of the module, with optimal yearly angle (Bayón Method)	VM4J_M1_D_A_Irrad.RT1_Avg	W/m^2
Total irradiance of table 1, frontal part of the module, with optimal angle proposed by Lorenzo	VM4J_M1_D_L_Irrad.RT1_Avg	W/m^2
Total irradiance of table 1, frontal part of the module, with optimal angle proposed by Jacobson	VM4J_M1_D_J_Irrad.RT1_Avg	W/m^2
Number of 2.5 µm particles in the air	PM2.5_Avg	µg/m^3
Number of 10 µm particles in the air	PM10_Avg	µg/m^3

As mentioned previously the software provided by SolarEdge to access the inverter's data already provides options to perform a good standard analysis. However, not all

features are useful for in-depth analysis and some of the features lack customization options. For the use intended for the software, the main interested lies in the energy being generated during each hour by each panel and the full string. The software also allows tracking the current and voltages of each module, however, that analysis is currently not being performed. The variables currently being monitored by the web system are below.

Table 3.5.- Variables measured by EdgeSolar Software

Parameter	Variable name	Units
Energy panel 1 (Tracker 1 PV1 module)	VM4SC_IT1_PV1_Ener.P1.1.1	Wh
Power panel 1 (Tracker 1 PV1 module)	VM4SC_IT1_PV1_Pote.P1.1.1	W
Energy panel 2 (Tracker 1 PV2 module)	VM4SC_IT1_PV2_Ener.P1.1.2	Wh
Power panel 2 (Tracker 1 PV2 module)	VM4SC_IT1_PV2_Pote.P1.1.2	W
Energy panel 3 (Tracker 1 PV3 module)	VM4SC_IT1_PV1_Ener.P1.1.3	Wh
Power panel 3 (Tracker 1 PV3 module)	VM4SC_IT1_PV1_Pote.P1.1.3	W
Energy panel 4 (Tracker 1 PV4 module)	VM4SC_IT1_PV1_Ener.P1.1.4	Wh
Power panel 4 (Tracker 1 PV4 module)	VM4SC_IT1_PV1_Pote.P1.1.4	W
Energy panel 5 (Tracker 1 PV5 module)	VM4SC_IT1_PV1_Ener.P1.1.5	Wh
Power panel 5 (Tracker 1 PV5 module)	VM4SC_IT1_PV1_Pote.P1.1.5	W
Energy panel 6 (Tracker 1 PV6 module)	VM4SC_IT1_PV1_Ener.P1.1.6	Wh
Power panel 6 (Tracker 1 PV6 module)	VM4SC_IT1_PV1_Pote.P1.1.6	W
Energy panel 7 (Tracker 1 PV7 module)	VM4SC_IT1_PV1_Ener.P1.1.7	Wh
Power panel 7 (Tracker 1 PV7 module)	VM4SC_IT1_PV1_Pote.P1.1.7	W
Energy panel 8 (Tracker 1 PV8 module)	VM4SC_IT1_PV1_Ener.P1.1.8	Wh
Power panel 8 (Tracker 1 PV8 module)	VM4SC_IT1_PV1_Pote.P1.1.8	W
Energy panel of the 8-module set (Tracker 1 String)	VM4SC_IT1_STR_Ener.Str.1.1	Wh
Power panel of the 8-module set (Tracker 1 String)	VM4SC_IT1_STR_Pote.Str.1.1	W



3.2.- Data cleaning

As mentioned earlier in the chapter, the non-optimal conditions of the environment necessitate the design of a structure capable of accommodating measurement failures. In order to standardize these raw files generated by the datalogger, a program in Python was developed, bearing in mind the necessities of the requirements. In this section, the main focus will be regarding the files obtained from both dataloggers, as they are more error-prone, nevertheless, there are some parts of the pipeline that also need to be applied to those obtained from the inverter's software.

When dealing with raw data there are three main concerns:

- **Missing values**: when some records are missing, in this case, minutes or even hours where no new rows were added to the CSV file.
- **Incorrect measurements**: due to the precision of the sensors or some external factor, wrong or impossible measurements might be added to a certain row.
- **Corrupt data**: the file itself is corrupt due to a malfunction and cannot be opened.

The process of data cleaning in this project has a very specific goal, add the 1440 records expected each day to the data repository making sure no incorrect measurements are appended as well as creating 24-record files from a summary of them. In the **sections 3.2.1** and **3.2.2** this part process is expressed step by set. Also as it is going to be explained in **section 3.3**, once the previous goal is achieved, incorrect fixable measurements will get modified.

3.2.1.- File examination

To start off, the program must receive a file to process. The script is designed to check repeatedly on a folder on the lab computer it is running on for new raw CSV files. As shown in code fragment **3.1**, the program is always inside an infinite loop that sleeps for ten seconds if no file is found in the input directory. Once it finds a new file it will try and open it to check if it is a valid file and extract its header. The header must be read to differentiate which datalogger produced that specific log. The name the dataloggers give

to the file is set to ”KLOG” + a number. This numerical value represents the cumulative count of logs generated by the datalogger since its initial operation. Due to that particular circumstance, the filename alone is not enough to differentiate which datalogger the file comes from, therefore not allowing to later check whether that day’s data has been parsed already.

Code fragment 3.1.- Infinite loop to check for CSV files

```
while True:
for file in os.listdir(PATH_INPUT_FILES):
    if file.lower().endswith('.csv'):
        file_to_process = os.path.join(PATH_INPUT_FILES, file)
        main(file_to_process)
time.sleep(10)
```

With that header and filename, a function in charge of checking if the file has already been parsed is called. The idea behind this is to reduce human error while extracting the raw files from the datalogger and copying them onto the computer. Thanks to the header, it is possible to know which datalogger created that file, and so with this information check the corresponding log of already parsed files to see if that particular CSV file has been processed already by the system.

It is important to note that, if the user wants to re-process a particular file, the program also gives him the chance to do so by asking via the terminal if he is willingly re-adding a file to the system to modify some previous data.

In case a human error was made, and the user does not want to re-process that file, the program deletes that original file from the input folder and continues to check if any more CSV files are available in that folder. If on the other hand, the file is new, or the user wants to process it again, the program continues its execution.

Code fragment 3.2.- Function to ask the user if he wants to re-process a file

```
def ask_to_continue(filename_original):
    user_input = input(f"The file {filename_original} has already been parsed. "
                       f"Do you wish to continue and rewrite the data? [y/n] ")
    while user_input not in ["y", "n"]:
```

```
        print("Invalid input. Please enter 'y' or 'n' ")
        user_input = input(f"The file {filename_original} has already been parsed. "
                           f"Do you wish to continue and rewrite the data? [y/n] ")
    if user_input == "y":
        return True
    else:
        return False
```

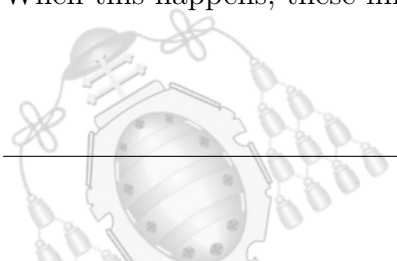
The next step in the operation is to format the first two lines to leave just the relevant information and rename the file for better human understanding. With the extracted header from the previous step and the original filename, the file is copied into a folder where all the processed files are stored. The new name of this file for storage will be the addition of its original name and "Data", if it comes from the second datalogger or "DataEstaci" if it comes from the first one. The name of the folder is **"RawDataParsed"**.

The following verification to be made is if the file has any records, or is it just composed of the variable's names. If this file is empty, an error message is returned and it goes back to the looped execution to check for new files. An error message is also printed to alert the user.

3.2.2.- Cleaning the archives

At this point, it has been verified that the file is uncorrupted and has some records. It is now time to start going through the file to see if all the measures have been taken and recorded correctly.

The first verification to be made is the number of records. As both dataloggers produce a log from each sensor every minute, and it generates one file every day, a correct file should have 1440 rows. **Function 3.3** shows how this operation is performed. While loading the CSV file passed as an argument to a pandas data-frame, an extra check is performed. Due to a malfunction of the device, it is possible that a row has more columns than expected. When this happens, these lines must be skipped to have a consistent row size.



The function shown in **3.3** has two possible outcomes. If everything is correct and the file has 1440 rows, it is copied onto the folder **"TratarDatos"**. This folder stores the files, without any automatic changes made to them. In the opposite case, when there are less than 1440 records, the file is copied onto **"RevisarDatos"**. Inside this folder, the incomplete files can be stored to manually check what errors occurred and perhaps try and find a solution.

Code fragment 3.3.- Function to check if a file has 1440 records

```
import shutil
import pandas as pd

def check_1440_records(file_path, output_ok_directory, output_error_directory):
    df = pd.read_csv(file_path, on_bad_lines='skip')
    if len(df) == 1440:
        file_name = os.path.basename(file_path)
        output_path = os.path.join(output_ok_directory, file_name)
        shutil.copyfile(file_path, output_path)
        print(f"[OK] The file {file_name} has 1440 records.")
        return True
    else:
        file_name = os.path.basename(file_path)
        output_path = os.path.join(output_error_directory, file_name)
        shutil.copyfile(file_path, output_path)
        print(f"[ERROR] The file {file_name} has some missing records")
        return False
```

The Boolean value returned by function **3.3** is evaluated afterwards on the main function as now it must decide the succeeding action to be taken. Assuming everything was OK and the function returned true, a function called **"datetime_together(path, name)" (3.4)** is invoked. This method takes as arguments the path of the file stored in folder **"RawDataParsed"** and a new filename.

This new filename is the result of the addition of the date of the records inside and "DataEstaci" or "Data" following the same criteria as previously. On following updates

of the system, the data coming from Arganda's dataloggers will have the same name but substituting "Data" or "DataEstaci" for "Arganda".

Code fragment 3.4.- Function to join columns Date and Time

```
def datetime_together(path_input_data, new_file_name):
    df = pd.read_csv(path_input_data)
    df['Datetime'] = pd.to_datetime(df['Date'] + ' ' + df['Time'], dayfirst=True)

    # Format the Datetime column
    df['Datetime'] = df['Datetime'].dt.strftime('%Y-%m-%d %H:%M:%S')

    df.set_index('Datetime', inplace=True)
    # Drop 'Date' and 'Time' columns
    df = df.drop(['Date', 'Time'], axis=1)

    df.reset_index(inplace=True)
    df.replace('---', 0.000, inplace=True) # Replace non taken measures to 0.
    path_treat_data = os.path.join(PATH_TREAT_DATA, new_file_name)
    df.to_csv(path_treat_data, index=False)
```

Once all these modifications are done, the program writes a new file into **"TratarDatos"** and deletes the previous file in this folder with the original columns **Date** and **Time** separated.

On the contrary, if there were less than 1440 records present on the file, the method called is **"revision(path, name)"**. The file that this function reads and loads into a data frame comes from the previously mentioned folder **"RevisarDatos"**. The goal this subroutine aims to achieve is to output a file with 1440 records without any external human intervention. Another important requirement is that the least amount of valid information should be discarded. Lastly, aim to prevent the generation of new data that could potentially offer misleading information due to insufficient referencing or estimation. To fulfil this, three use cases were defined:

- **Missing intermediate data**: when the datalogger or the sensors had a temporary malfunction that resulted in some minutes or even hours not recording any data.

- **Missing data until one point**: the datalogger has not been working properly from the previous day and therefore is missing data from the start of the day until the problem got fixed.
- **Missing data from one point onwards**: the datalogger stopped working at a specific moment in the day of the file that is being parsed. No more rows from that point.

The three use cases are non-mutually exclusive and the code in charge of taking care of this problem should be able to deal with more than one at the same time. It is because of this that the same function will be in charge of checking if the goal of the method has been achieved. This way no matter the circumstances the output will be consistent. The procedure to solve each of those use cases is the following:

1. Once the CSV file is loaded into a data frame, all the rows but the first one get selected. These rows now get re-sampled with a frequency of one minute to ensure that all missing instances of the day have a record. Then these re-sampled rows get concatenated with the initial measurement. After that, all rows but those of type *"datetime64"* get cast to floats. When this is done, a linear interpolation [16] is performed to fill the Not a Number (NaN) values on the newly created rows.
2. Afterwards, the NaN remaining; those from columns where a sensor did not measure in all day due to a failure, get filled with zeroes. And only then do all values in the data frame get rounded to three decimal places, the same as the precision of the sensors.
3. With the first use case solved, the focus shifts to the other two. When the values missing are from the start or the end of the data frame, re-sampling will not create the rows missing. It is because of that, the first time and the last time of the day must be forcefully set by the program.
4. By creating a new data frame with a value per minute from the start time to the end time, an empty data frame with 1440 rows is obtained. By merging this data frame with the original one, a complete data frame is formed.

5. In this case the values that remain NaN are not replaced by zeroes because that would mean estimating values at times when no valid reference to do so exists.
6. Finally, this data frame with 1440 records is saved in a new CSV file on "TratarDatos".

Code fragment 3.5.- Automatic revision of missing rows

```
def revision(path_raw_data, new_file_name):
    df = pd.read_csv(path_raw_data, on_bad_lines='skip').dropna()

    # Format the Datetime column
    df['Datetime'] = pd.to_datetime(df['Date'] + ' ' + df['Time'], dayfirst=True)
    df.set_index('Datetime', inplace=True)

    first_row = df.head(1) # First row does not follow the pattern
    remaining_rows = df.iloc[1:]
    # Resample the remaining rows using a time-frequency of 1 minute
    resampled_rows = remaining_rows.resample('1T').asfreq()

    # Concatenate the first row with the resampled rows
    new_df = pd.concat([first_row, resampled_rows])
    # Drop 'Date' and 'Time' columns
    new_df = new_df.drop(['Date', 'Time'], axis=1)
    new_df.replace('---', 0.000, inplace=True) # Replace non taken measures to 0.

    cols_to_cast = new_df.select_dtypes(exclude=['datetime64']).columns
    new_df[cols_to_cast] = new_df[cols_to_cast].astype(float)
    # Interpolate missing values
    new_df = new_df.interpolate(method='linear')
    new_df.fillna(0.0, inplace=True) # Cleaning empty cells
    new_df = new_df.round(decimals=3) # Round to 3 decimal places
    # Reset index to Datetime
    new_df.reset_index(inplace=True)

    # Missing the first or last few measures
    if len(new_df) != 1440:
        # Get the start and end time of the day
```

```
    start_time = new_df['Datetime'].min().replace(hour=0, minute=0, second=0,
        microsecond=0)
    end_time = new_df['Datetime'].max().replace(hour=23, minute=59, second=0,
        microsecond=0)

    # Create a new data frame with one row per minute of the day
    index = pd.date_range(start=start_time, end=end_time, freq='T')
    empty_df = pd.DataFrame(index=index)

    # Merge the new DataFrame with the original DataFrame
    new_df = pd.merge(empty_df, new_df, left_index=True, right_on='Datetime',
        how='outer')

    new_df.set_index('Datetime', inplace=True)
    # Reset index to Datetime
    new_df.reset_index(inplace=True)

# Save the output DataFrame to a CSV file
path_treat_data = os.path.join(PATH_TREAT_DATA, new_file_name)
new_df.to_csv(path_treat_data, index=False)
```

3.3.- Treating the data

Thus far the data cleaning of the input files has primarily focused on missing or unusable data. This is a very important process in data cleaning, but it would not be wise to stop there. Thanks to powerful Python libraries such as Pandas [20] or Suntime [21], there are still many ways in which the input files can get depurated before concatenating their content into the data repository file.

To avoid errors adding up while calculating energy generated, before sunrise and after sunset, irradiance values must be set to zero. Some sensors may still give very small values during these hours, however, this data is useless as no power is actually being generated. Another possible problem that the data may have is the fact that at night, irradiance sensors can give negative results. This situation is physically impossible, making it illogical to store such values. If this were to happen, the appropriate action would be to set the

corresponding value to zero. Code fragments **3.6** and **3.7** show how all of the operations previously mentioned are performed.

Code fragment 3.6.- Function treat the data

```
def treating_data(filename):
    basefilename = os.path.basename(filename)
    df = pd.read_csv(filename)

    # Irradiance measurements columns
    irradiance_cols = [col for col in df.columns if 'Irrad' in col]
    df.replace('---', 0.0, inplace=True) # Replace non taken measures to 0.
    df[irradiance_cols] = df[irradiance_cols].apply(pd.to_numeric)

    # Set negative values in irradiance_cols to 0
    for col in df.columns:
        if col in irradiance_cols:
            df[col] = df[col].apply(lambda x: 0 if x < 0 else x)

    # Convert 'DateTime' column to datetime data type
    df['Datetime'] = pd.to_datetime(df['Datetime'])
    # Get the date (day) of the first value
    date = df['Datetime'].dt.date.iloc[0]
    sunrise_t, sunset_t = get_sunrise_sunset(date)

    # Set irradiance_cols values to 0 before sunrise_t and after sunset_t
    for col in irradiance_cols:
        df.loc[df['Datetime'] < sunrise_t, col] = 0
        df.loc[df['Datetime'] > sunset_t, col] = 0

    filename1440 = basefilename.split('Data')[0] + 'Data1440' +
        basefilename.split('Data')[1]
    output = os.path.join(PATH_DATA_GIJON_1440, filename1440)
    df.to_csv(output, index=False)

    return output
```

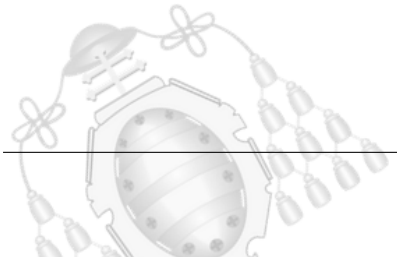
Code fragment 3.7.- Function to get the sunrise and sunset time at Gijón

```
from suntime import Sun, SunTimeException

def get_sunrise_sunset(date):
    # Coordinates of Gijon
    latitude = 43.5359
    longitude = -5.6619
    sun = Sun(latitude, longitude)
    try:
        # Get the sunrise and sunset times for the given date
        sunrise = sun.get_sunrise_time(date)
        sunset = sun.get_sunset_time(date)

        # Format the sunrise and sunset times
        sunrise_time = sunrise.strftime("%Y-%m-%d %H:%M:%S")
        sunset_time = sunset.strftime("%Y-%m-%d %H:%M:%S")
        return sunrise_time, sunset_time

    except SunTimeException:
        return None, None
```

3.3.1.- Re-sampling for twenty-four measurements

Even though the files with 1440 records are finally parsed, and ready to be added to the data source, one more process is going to be done with the document before moving on to the next one, to transform the measures from one per minute to one per hour. This will result in a file with 24 records and a few modified variable names and magnitudes

As explained in **section 2.4** probably the most relevant variables recorded on both dataloggers are those regarding solar irradiance. Measurements stored on variables with the word "Irrad" on them are representing the average value of the solar irradiance over that minute. The measurement is expressed in units of Watts per square meter, as demonstrated in the tables labelled as **Table 3.3** and **3.4**.



Solar irradiance is often integrated over a period of time, for example, an hour, to calculate solar irradiation. This magnitude represents the solar energy generated in a given period. It is usually measured in Wh/m^2 or J/m^2.

During the process of resampling the dataset from a frequency of one data point per minute to one data point per hour, the energy generated within each time interval is determined using the rectangle rule **(3.1)**. Given that measurements are captured every minute, calculating the energy generated within an hour follows a similar approach to computing the mean value. Similarly, for other variables undergoing resampling, their mean values can be computed conventionally to yield a meaningful representation for the respective hour. Consequently, as depicted in function **3.9**, the new data frame is populated with the mean values corresponding to each hour.

$$\int_a^b f(x)\,dx \approx (b-a)\cdot f\left(\frac{a+b}{2}\right) \tag{3.1}$$

This part of the process is key, as having this other way of seeing the data allows for a more varied analysis. In this case, it is of even bigger importance because the data coming from the inverter is going to come in terms of the energy generated by each module each hour it was active.

The script in charge of managing the inverter data does not need to do as many corrections to the data. Nonetheless, as it only creates logs when there is power coming from the panels, the number of rows is very variable. In order to standardize the number of rows added each day to the repository, the data frame is resampled to obtain one log every hour every day and fill with zeros that extra records. **Function 3.8** shows how these modifications are done in Python.

Code fragment 3.8.- Function force 24 energy records each day

```
def resample_24(filepath):
    df = pd.read_csv(filepath, parse_dates=['Time'], index_col='Time')
    df = df.resample('H').asfreq()
    # Create a date range from 1 AM of the first day to 12 AM of the next day
    start_date = df.index[0].replace(hour=1, minute=0, second=0)
```

```
end_date = start_date + pd.DateOffset(days=1) - pd.DateOffset(hours=1)
date_range = pd.date_range(start=start_date, end=end_date, freq='H')

# Reindex the dataframe with the new date range
df = df.reindex(date_range)
df.index.name = 'Time'
df = df.fillna(0)
df.to_csv(filepath)
```

3.3.1.1.- Time-zone synchronization

Synchronizing timezone measurements between comparable rows on two different CSV files is of utmost importance when conducting data analysis or comparisons. In the realm of data processing, especially when dealing with multiple data sources or systems, ensuring that the timestamps align correctly is crucial for accurate and meaningful results. Timezone discrepancies can lead to inconsistencies and erroneous conclusions, potentially impacting the reliability and validity of the analysis.

As it has been stated previously in **section 3.1**, the dataloggers mark each measurement with the current UTC time, but the inverter does so with the local time, in this case, Spanish time. In consequence, one of the measurements must be translated into the other's time zone.

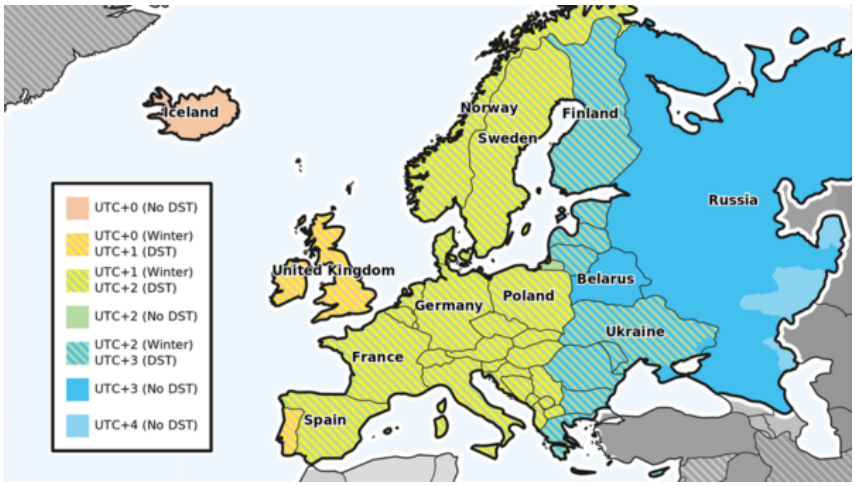


Figure 3.8.- Timezones Europe

To simplify the posterior analysis, it was decided that the data from the sensors will get shifted to local time. Either option had its pros and cons, but as both posed a similar difficulty it was decidede to have all measurements in local time so it was more intuitive for following monitoring. Code fragment **3.9** shows the function in charge of reading the file with a record per minute in UTC time and outputting a new file using the local time as reference and with the energies calculated as expressed in **3.3.1**.

It is important to also note that a similar function to **3.9** is called just before, and it also re-samples the log file with 1440 records but then outputs a file with the UTC time.

Code fragment 3.9.- Function to get 24 records per day on local time

```
def resample_24HLocal(path_file_1440):
    df = pd.read_csv(path_file_1440)
    # Convert 'Datetime' column to datetime type
    df['Datetime'] = pd.to_datetime(df['Datetime'])
    # Set 'Datetime' column as the index
    df.set_index('Datetime', inplace=True)

    # Set index to UTC timezone
    df = df.tz_localize('utc')
    # Shift time to Madrid
    df = df.tz_convert('Europe/Madrid')

    # Resample to 1 record per hour and calculate the mean of each hour
    df_resampled = df.resample('H').mean()
    df_resampled = df_resampled.round(decimals=3)
    # Reset the index to have 'Datetime' as a column again
    df_resampled.reset_index(inplace=True)
    # Remove the timezone from the column
    df_resampled['Datetime'] = df_resampled['Datetime'].dt.tz_localize(None)

    basefilename = os.path.basename(path_file_1440)
    filename24 = basefilename.split('Data')[0] + 'Data24Local' +
        basefilename.split('Data')[1]
    output = os.path.join(PATH_DATA_GIJON_24HORALOCAL, filename24)
    # Save the resampled data to a new CSV file
```

```
    df_resampled.to_csv(output, index=False)
    return output
```

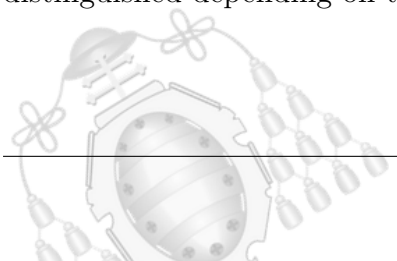
3.4.- Storing the data

The file is now ready to be stored in its permanent location where it will be consumed by the web application. But first, it must go through a final step.

In order to access the data and also to maintain all records in the same file, the parsed file now will be appended to the file used as a repository for all records. As there are two different data sources and two different time interval measurements, there will be four different CSV files in charge of acting as the data source for the web application. Two for the data generated by the first datalogger, one with data on UTC time and a log per minute, and another with local time and a log per hour. In the case of the second datalogger, the same logic is followed. These files will be called **"Datalogger1.csv"**, **"Datalogger24hLocalEstaci.csv"**, **"Datalogger2.csv"** and **"Datalogger24hLocal.csv"**.

The idea behind having two different repositories with repeated data, as the 24-record files are just a partial view of the original 1440 entries per day, is to allow for analysis from two different perspectives. If the objective is to analyze the potential of the solar installation, more records per minute allow for a deeper study with higher sensibility. However if, as it is more likely, the goal is to measure the efficiency of the cells and ensure the correct maintenance of all the components, the more useful insights will come from the comparison between the measured theoretical energy and the obtained energy directly acquired from the string inverter.

Because the variables measured each day may differ, the automatic appending of the records cannot be done carelessly. As shown in code fragment **3.10** a number of considerations are made before concatenating the new records. Three cases can be distinguished depending on the nature of the new data about to be added:



- New record with same or fewer variables: the simplest case possible. The new rows will simply get concatenated at the end of the file and every variable-value pair has a column to put these data in. Those columns where no measurement was taken will be left as a NaN.
- New record with more variables: if a new measure that previously was not being monitored is present on the file, the column must be created and filled with NaN values for the previous records that did not include it.
- Old record being re-processed: a file that is purposefully being reprocessed is going to be added to the data source where records of that specific day and time are already in place. As it is assumed that this new data is useful, the new rows must replace the old ones. This is performed by removing duplicate records and prioritizing the last occurrence.

Code fragment 3.10.- Function to append new records from the dataloggers

```
def push_into_db(filename, datalogger_db):
    new_data = pd.read_csv(filename)
    database = pd.read_csv(datalogger_db)

    # Check if the new file has more columns than the database
    new_columns = set(new_data.columns) - set(database.columns)
    if new_columns:
        # Add the new columns to the database and set values to 0 for previous rows
        for column in new_columns:
            database[column] = np.nan

    # Append the new data to the database, overwriting rows with the same 'Datetime'
    database = pd.concat([database, new_data]).drop_duplicates(subset='Datetime',
        keep='last')
    database.sort_values(by='Datetime', inplace=True)

    # Delete columns with no values
    database.dropna(axis=1, how='all', inplace=True)
    print(f"File: {os.path.basename(filename)} has been added to database\n")
    # Save the updated database back to the CSV file
    database.to_csv(datalogger_db, index=False)
```

For the data coming from the inverter, the process is similar but has some key differences that make the function that pushes all the data onto the repository of records unique. As it can be seen on **code fragment 3.11** before it adds any data, it must check if a record has already been added at that time. In the case in which a record is completely new, it simply concatenates its new information. On the other hand, if the data is from the same date, the results must be combined.

The approach described here significantly differs from the one employed in **function 3.10**. In this particular scenario, the objective is not to overwrite the information but rather to merge it. The rationale behind this choice stems from the data acquisition process, where the data is directly obtained from the SolarEdge software. While the software is user-friendly and provides a substantial amount of information, it has limitations in terms of column selection, which prompted the need for a different approach.

Due to this limitation, downloading the data from all the solar cells and the full string onto a single CSV file is not possible. To overcome this, a two-step approach can be employed. Firstly, the file containing the data from the entire string can be processed, followed by the processing of the data from each individual panel. This arrangement allows for storing the information on the same row, facilitating easier access in the subsequent stages of the monitoring system.

Code fragment 3.11.- Function to append new inverter energy and power data

```
def push_into_repo(filepath_to_push, tracker):
    new_data = pd.read_csv(filepath_to_push)
    database = pd.read_csv(paths_repos[tracker])

    overlapping_records = database[database['Time'].isin(new_data['Time'])]
    if overlapping_records.empty:
        database = pd.concat([database, new_data])
    else:
        database.set_index('Time', inplace=True)
        new_data.set_index('Time', inplace=True)
        # database = database.combine_first(dfs)
        database.update(new_data)
        database.reset_index(inplace=True)
```

```
# Order the database by 'Datetime' in ascending order
database.sort_values(by='Time', inplace=True)
database.to_csv(paths_repos[tracker], index=False)
```

All throughout this section, it has been stated where each function got its information and where this information was stored. However, to have a clearer picture of the whole file system here is the complete folder architecture shown in diagram 3.9.

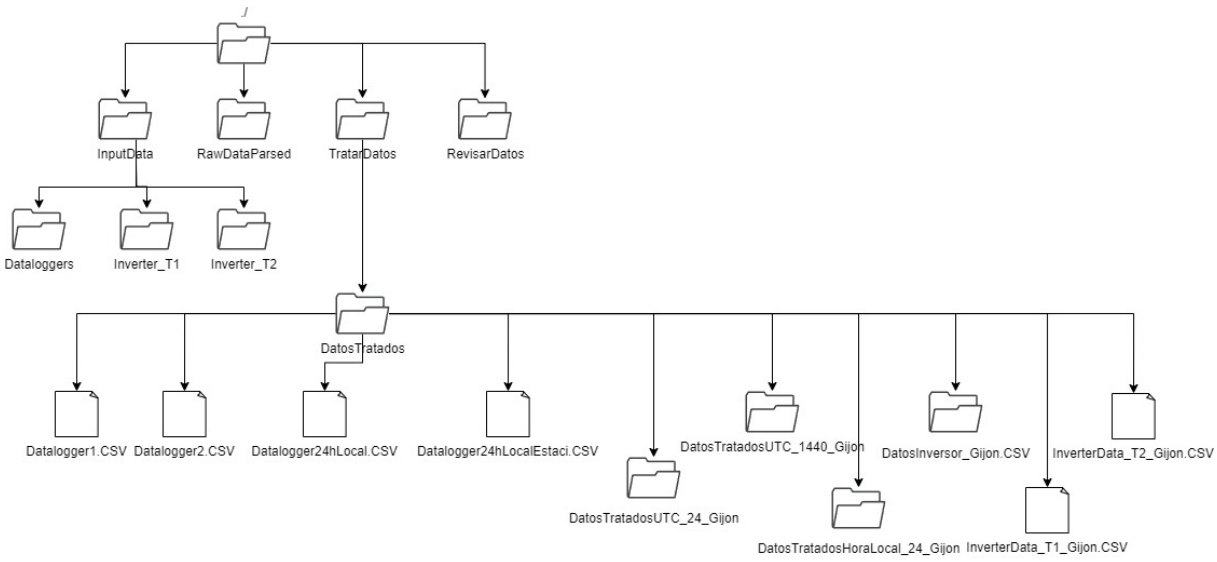


Figure 3.9.- Local file storage organisation

3.5.- Technologies used

In undertaking the tasks outlined in this chapter, numerous technologies were considered, recognizing the importance of aligning the chosen methods with the requisites, contextual factors, and potential future enhancements.

The goal of this section is to present the rationale behind choosing Python as the programming language, Pandas as the data manipulation library, and Jupyter Notebook as the interactive environment for efficient data processing and analysis. By examining the advantages and capabilities of each technology, it is possible to understand why this

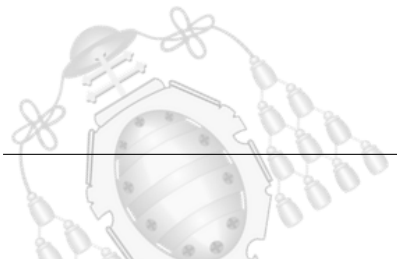
specific combination is well-suited for handling and extracting insights from PV panel sensor data.

3.5.1.- Python

Python [12] was chosen as the primary programming language for processing PV panel sensor data due to several key factors. Firstly, Python's simplicity and ease of use make it accessible to users of varying programming backgrounds, enabling efficient collaboration and knowledge sharing within the research community. Secondly, Python boasts an extensive collection of libraries tailored for data manipulation, analysis, and visualization, providing a wealth of pre-built functionalities that accelerate development and reduce implementation complexity. Finally, Python's clean and readable syntax promotes code legibility, making it easier to understand, maintain, and debug complex data processing pipelines for future features.

3.5.1.1.- Integrated Development Environment

The Integrated Development Environment (IDE) of choice to develop the script was PyCharm [13]. PyCharm offers a range of features that enhance the development experience, boost productivity, and support efficient coding practices. The IDE provides advanced code editing functionalities, including autocompletion, syntax highlighting, and error detection, which help developers write clean and error-free code. Additionally, PyCharm supports seamless integration with version control systems, such as Git to ensure code integrity. The interactive debugging capabilities of PyCharm enable developers to identify and resolve issues efficiently, further enhancing the reliability of the data processing workflow.



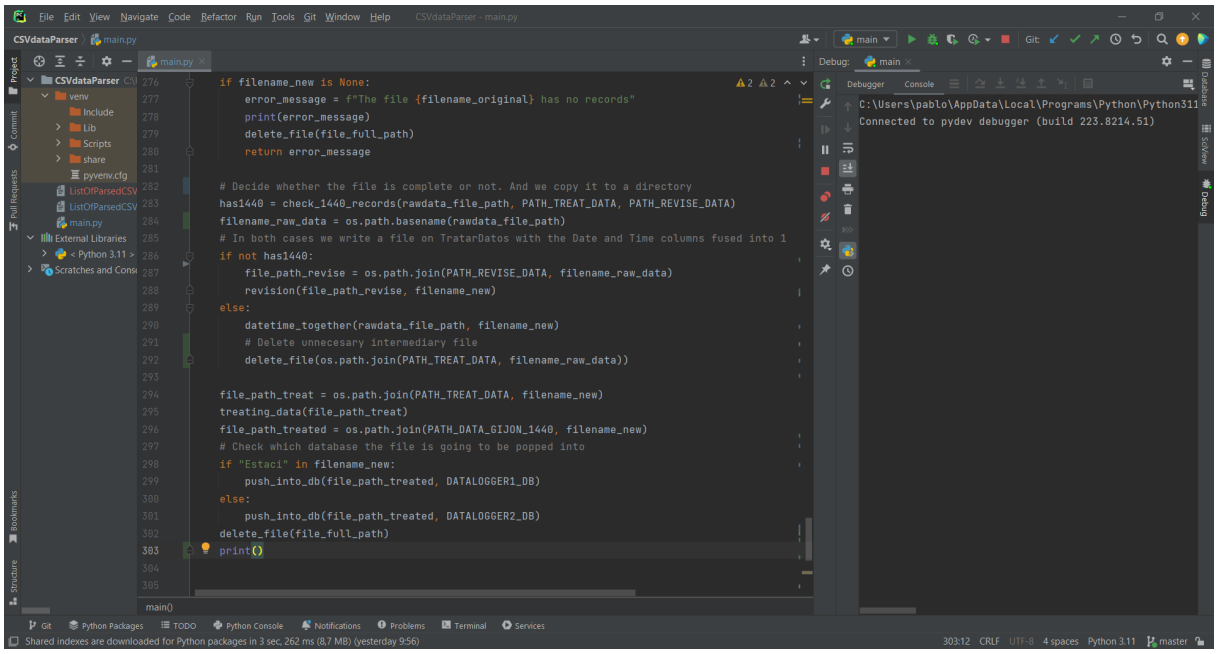


Figure 3.10.- Pycharm IDE. Debug mode

3.5.1.2.- Libraries

As Python is one of the most used programming languages in the field of data science and data analysis [19], there were a number of library options to choose from.

Among the various data manipulation libraries available, Pandas stands out for its versatility and powerful capabilities, making it an ideal choice for PV panel sensor data processing. Pandas offers a highly efficient and intuitive way to handle tabular data through its DataFrame object [14], which seamlessly handles CSV file loading, parsing, and manipulation. With Pandas, researchers and developers alike can effortlessly clean noisy or missing data, aggregate information, perform complex transformations, and filter data sets based on specific criteria. These versatile features empower users to gain valuable insights into their data. Pandas offers an array of functionalities that streamline the data processing workflow. It provides capabilities for data cleaning, such as handling missing values, removing duplicates, and transforming data types. Aggregation and grouping operations enable the calculation of descriptive statistics, averages, and other summary metrics. Furthermore, Pandas enables seamless data merging, concatenation, and reshaping for efficient integration of multiple data sources.

During the evaluation of libraries for processing the sensor data, the Polars library [22] emerged as an alternative to Pandas. Polars offers similar functionalities for data manipulation and analysis, with the added advantage of being designed for high-performance data processing. However, after careful consideration, it was decided to utilize Pandas as the primary library for several reasons. Firstly, Pandas has a larger and more established user base, providing a wealth of community support, resources, and documentation. Secondly, Pandas' extensive ecosystem of complementary libraries and tools makes it highly versatile for various data analysis tasks. Lastly, the familiarity and ease of use associated with Pandas among researchers and developers influenced the decision, as it enables simpler adoption and eases the learning curve. Overall, while Polars exhibits promising performance capabilities, the overall ecosystem, community support, and user-friendliness of Pandas ultimately led to its selection as the preferred library for this data processing pipeline.

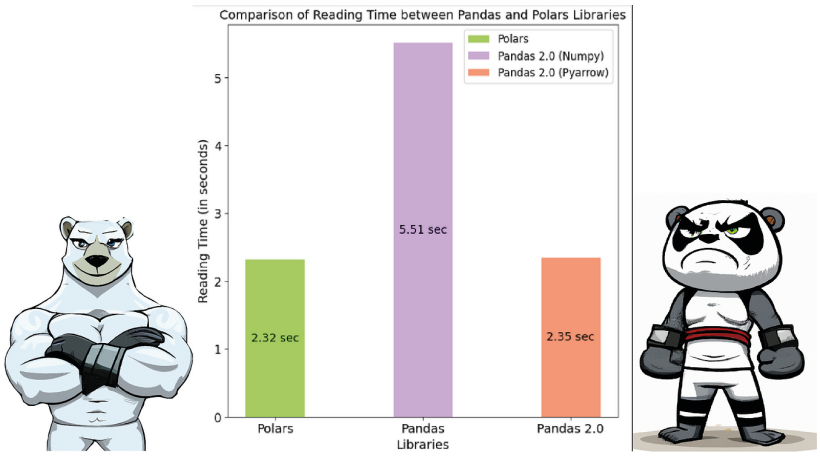


Figure 3.11.- Pandas vs Polars. Image obtained from [23]

3.5.2.- Jupyter Notebook

Jupyter Notebook [15] serves as an invaluable tool for individual developers engaged in processing sensor data, offering streamlined testing capabilities and facilitating easy visualization of resulting DataFrames. With its interactive and iterative nature, Jupyter Notebook allows developers to conveniently test functions and methods in real-time while viewing the modifications made to the DataFrames. This functionality is akin to unit

testing, enabling developers to verify the correctness of their code and ensure the expected transformations and manipulations are accurately applied.

By executing code cells within Jupyter Notebook, developers can observe the effects of various operations, such as filtering, sorting, and data transformations, on the DataFrames instantly. This real-time feedback enables quick identification of any issues or discrepancies, improving the efficiency of the development process. Developers can iteratively refine their code and validate the modifications made to the DataFrames, ensuring the data processing pipeline functions as intended. In **figure 3.12** an example of some of the tests performed before adding the code to the main script is shown.

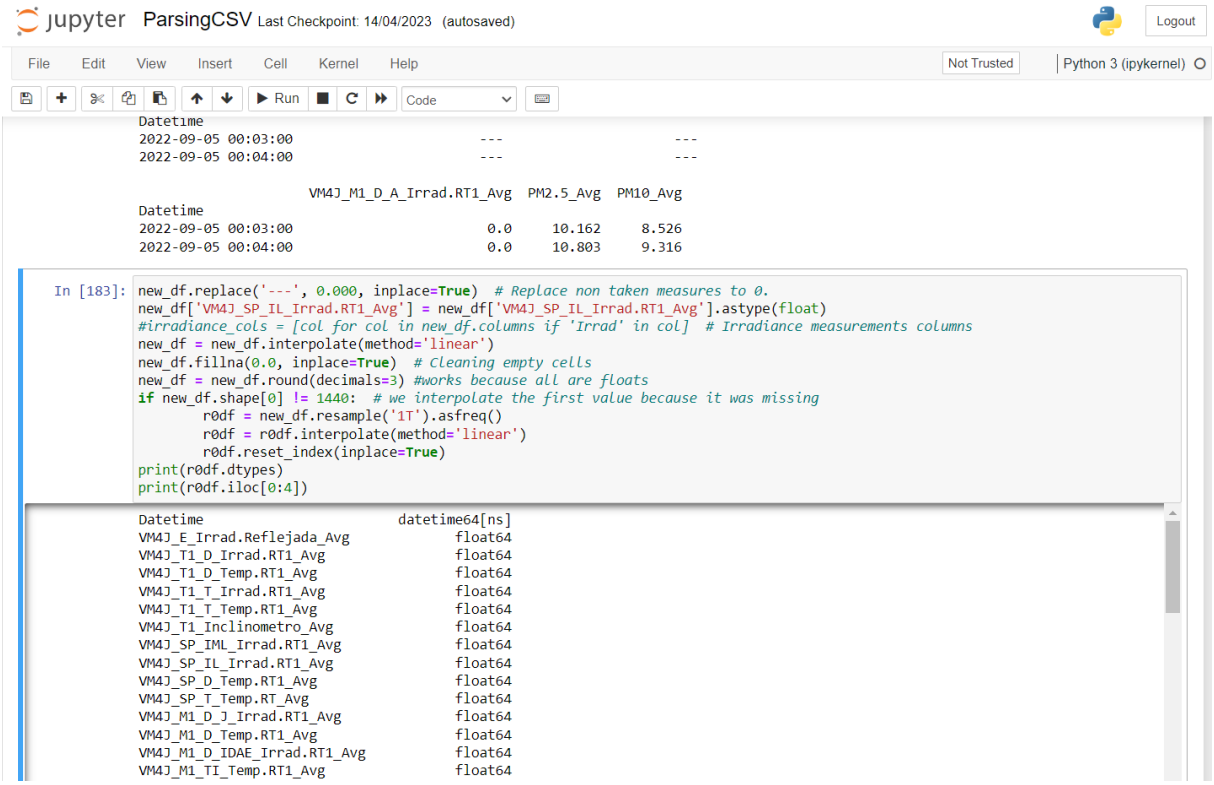


Figure 3.12.- Use case of Jupyter Notebook

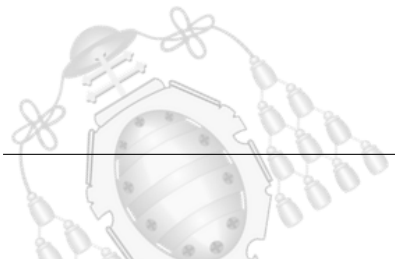
Moreover, Jupyter Notebook's ability to integrate visualizations, such as plots and charts, allows developers to gain deeper insights into the transformed data. Visual representations of the modified DataFrames aid in identifying patterns, trends, and anomalies, thereby facilitating data analysis and quality assessment. This visualization

aspect further enhances the testing process, enabling developers to visually confirm that the expected modifications have been correctly applied to the data.

In essence, Jupyter Notebook facilitates working on PV panel sensor data processing to test functions and methods efficiently, visually validate DataFrame modifications, and iteratively refine the code. By providing an interactive and visual environment, Jupyter Notebook not only supports the development process but also encourages thorough unit testing practices. This integrated testing and visualization capability ultimately improves the reliability and accuracy of the data processing workflow, contributing to the overall quality of the research outcomes.

3.5.3.- Excel

Excel was selected as the tool for examining CSV files generated by the data-cleaning script in this engineering paper for several reasons. First and foremost, Excel provides a familiar and user-friendly interface, making it accessible to a wide range of users with varying technical backgrounds. Its widespread availability and ease of use make it a practical choice for quickly visualizing and exploring data. Furthermore, Excel offers various data manipulation and analysis functionalities, allowing for simple calculations, filtering, and sorting operations. Additionally, Excel's graphical capabilities enable the creation of charts and graphs, aiding in the visual representation of raw and cleaned data from the PV panels. Overall, Excel proves to be a versatile and efficient tool for data inspection and initial analysis in the context of cleaning data from PV panels.



4. Web development

4.1.- Frontend

In order to fulfil the goals presented in this project, a key element to design was the website's interface. For this task, it was decided to use a TypeScript-based, free and open-source single-page web application framework from Google called Angular. The reasoning for this decision is explained in **section 4.3.** It is used in conjunction with Angular Material and Bootstrap for better user experience and functionality.

The web application is built from components, that interact with each other to create a good user experience. As the functionalities of the project were well-defined from the get-go, the design didn't suffer many changes during its implementation. The requisites for this website were:

- **Authenticate users**: to only allow authorized people to access the data.
- **Download data**: in CSV format from different sources selecting the desired rows (dates) and columns (magnitudes).
- **Watch data evolution**: on different graphs, grouped by magnitude.
- **Compare dates graphically**: two different time periods, same graphs on both sides of the screen.
- **Documentation available online**: of the different components of the solar plant.

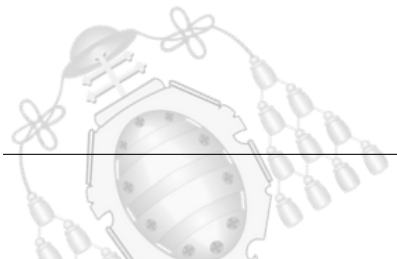
Apart from these features, the web application should be designed in such a way that can be easily updated for newer sources of data, as well as for newer features that may be needed in the future. Angular is ideal for this, as its modular approach to web development allows for the reuse of components, services and modules for newer extended features. This becomes of vital importance as the number of strings in the solar park increases and even newer installations on newer locations become available for monitoring using the same system.

4.1.1.- User authentication

Authentication and authorization are two words commonly used together and sometimes misunderstood. When somebody authenticates themselves, that means that they are proving they are who they say they are. Meanwhile, authorization is the function of specifying access privileges to some resources. The objective of the login component is to solve both of these processes and stop unauthorized users from accessing the data.

From a web-development standpoint, the implementation of such features requires the creation of a form where an email address and password are provided. The email and password details are to be kept confidential and solely utilized for accessing the designated application. Upon providing a correct email and password combination, the user's identity can be established, and their eligibility for resource access can be determined. If access is granted, the user will be redirected to a separate page where they can initiate the download of data in CSV format **(4.1.2)**.

It could be said that the website has two different states, differentiated by the privileges the user has been granted. The initial state blocks all tabs but the login one, and the only action the user can perform is to log into the app. On the navbar, the last element shows that you must log in before accessing any other tab. If you try to access a different component, by modifying the Uniform Resource Locator (URL) or by clicking on an element of the navbar, you will get redirected to the login page. However, if the login has been successful, the last element on the navigation bar changes into a logout button that specifies the email of the user currently using the site. In this second state, the user may access all functionalities and can also, when he is done, log out to end his session.



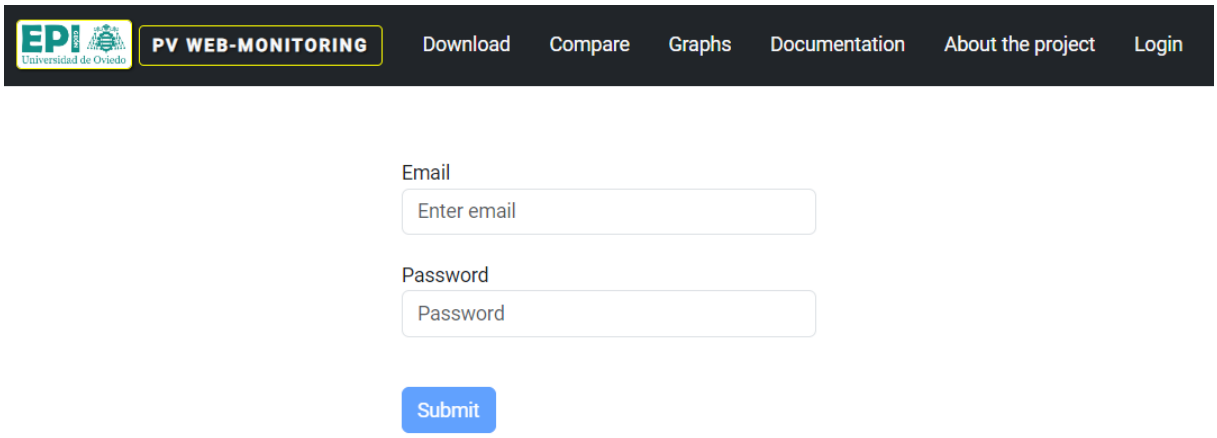


Figure 4.1.- Login page

With those things in mind, on the web application, one component and one service were developed to handle these tasks.

In Angular, a component is a fundamental building block of an application's user interface. It represents a self-contained, reusable unit that encapsulates both the User Interface (UI) and the associated logic. Components are responsible for rendering a specific part of the application's view and handling the interactions with that view.

The code fragment listed below shows the Hypertext Markup Language (HTML) in charge of the login phase. It is composed of a form element, inside some div containers, that fits two different inputs, an email and a password. The two inputs utilize the directive **formControlName** to sync a FormControl in an existing FormGroup. This then enables the use of validators to check if the email is in fact an email and if the password is at least six characters long. If those two requirements are met, the submit button is enabled so you can submit your credentials.

Code fragment 4.1.- login.component.html

```
<div class="container">
  <div class="row justify-content-center">
    <div class="col-md-6">
      <form [formGroup]="formLogin" (ngSubmit)="onSubmit()">
        <div class="form-group">
```

```
          <label for="email">Email</label>
          <input type="email" class="form-control" id="email" placeholder="Enter
              email" formControlName="email">
        </div>
        <div class="form-group">
          <label for="password">Password</label>
          <input type="password" class="form-control" id="password"
              placeholder="Password" formControlName="password">
        </div>
        <button type="submit" class="btn btn-primary"
            [disabled]="formLogin.invalid">Submit</button>
      </form>
    </div>
  </div>
</div>
```

When the submit button is pressed, it calls a function within the **LoginComponent** named **onSubmit()**. This function calls the login function on the **UserService** class as a promise and if the response is positive, it navigates to the download page as explained previously. If there was a problem, it is caught and an error message will appear in the form of a SnackBar [26].

Code fragment 4.2.- login.component.ts

```
import { Component} from '@angular/core';
import { Router } from '@angular/router';
import {FormControl, FormGroup, Validators} from '@angular/forms';
import { UserService } from '../services/user.service';
import { MatSnackBar } from '@angular/material/snack-bar';
@Component({
  selector: 'app-login',
  templateUrl: './login.component.html',
  styleUrls: ['./login.component.css']
})
export class LoginComponent{

  formLogin: FormGroup;
```

```
  constructor(private userService: UserService, private router: Router, private
      _snackBar: MatSnackBar) {
    this.formLogin = new FormGroup({
      email: new FormControl('', [Validators.required, Validators.email]),
      password: new FormControl('', [Validators.required, Validators.minLength(6)])
    });
  }
  openSnackBar(message: string, action: string) {
    this._snackBar.open(message, action);
  }
  onSubmit() {
    this.userService.login(this.formLogin.value)
      .then(response => {
        this.router.navigate(['/download']);
      })
      .catch(err => this.openSnackBar('Authentication error. Try Again', 'Close')
      );
  } ;
}
```

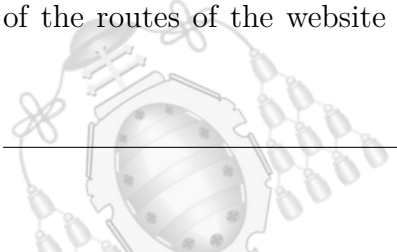
Services are classes responsible for organizing and providing functionality that can be shared across multiple components or modules within an Angular application. They facilitate the separation of concerns and promote code reusability. Services in Angular typically encapsulate data manipulation, business logic, Application Programming Interface (API) calls, and other operations that are not directly related to the user interface. They act as intermediaries between components and external data sources, such as servers, databases, or other services.

The **UserService** defined below provides the functions needed to log in and log out of the application as well as a subscription to the authorization state to be able to utilize it throughout the application. The methods that each function calls are imported from **@angular/fire/auth** module [24], which is part of the AngularFire library. Firebase is a cloud-based app development platform backed by Google which provides a variety of features, including authentication, that interacts very seamlessly with Angular. Its configuration and set-up will be explained on **section 4.2**.

Code fragment 4.3.- user.service.ts

```
import { Injectable } from '@angular/core';
import { Auth, signInWithEmailAndPassword, signOut, onAuthStateChanged } from
    '@angular/fire/auth';

@Injectable({
  providedIn: 'root'
})
export class UserService {
  userEmail: string null = null;

  constructor(private auth: Auth) {
    this.subscribeToAuthState();
  }

  login({ email, password }: any) {
    return signInWithEmailAndPassword(this.auth, email, password);
  }

  logout() {
    return signOut(this.auth);
  }

  private subscribeToAuthState() {
    onAuthStateChanged(this.auth, (user) => {
      this.userEmail = user ? user.email : null;
    });
  }
}
```

Finally, to redirect unauthorized users to the login component, inside the routing module some modifications need to be made. As it was shown before, using an imported module from angular-fire can help turn complex tasks like implementing an AuthGuard into a single line of code. With the provided **canActivate** helper and spread syntax the rest of the routes of the website can be protected against unauthorized users. As shown in



code fragment 4.4, the syntax is very legible and easy to modify and update for newer components.

Code fragment 4.4.- Routing configuration

```
import { canActivate, redirectUnauthorizedTo } from '@angular/fire/auth-guard';

const routes: Routes = [
  {path: '', component: DownloadCSVComponent,
  ...canActivate(()=>redirectUnauthorizedTo(['/login']))},
  {path: 'download', component: DownloadCSVComponent,
   ...canActivate(()=>redirectUnauthorizedTo(['/login']))},
  {path: 'compare', component: CompareComponent,
   ...canActivate(()=>redirectUnauthorizedTo(['/login']))},
  {path: 'graphs', component: GraphsComponent
  , ...canActivate(()=>redirectUnauthorizedTo(['/login']))},
  {path: 'about', component: AboutUsComponent
  , ...canActivate(()=>redirectUnauthorizedTo(['/login']))},
  {path: 'documentation', component: DocumentationComponent,
   ...canActivate(()=>redirectUnauthorizedTo(['/login']))},
  {path: 'login', component: LoginComponent}
];
```

One more security aspect to protect the website comes from the server used to host the network. As it will be further explained in **section 5**, everything is hosted on a computer within the university's network, which restricts access exclusively to users connected to this network. Consequently, individuals seeking to connect to the website must either be physically present within the university premises or establish a secure Virtual Private Network (VPN) connection to the university's network. This arrangement serves as an additional layer of security, leveraging the network management capabilities inherent to the university's infrastructure.

By enforcing this access limitation, the website benefits from the inherent security measures implemented by the university's network management. The network administrators employ various protocols, firewalls, and security measures to safeguard the internal network from unauthorized access and potential threats. This controlled

access ensures that only authorized individuals, either within the university or via VPN, can interact with the website, thereby enhancing data protection and minimizing the risk of external unauthorized access or malicious activities.

4.1.2.- Downloading data

The core functionality of the application. To perform a mathematical in-depth analysis of all the data generated by the multiple sensors installed, the user must be able to download the desired data with complete control of the dates and magnitudes to download. The **"DownloadComponent"** component and **"ConsultasService"** are in charge of this task.

As stated in the introduction to this chapter, a vital functionality the website should have is to download CSV files from one date to another. It must also be able to pick from which data source he wants to access the data and which magnitudes he is interested in. For this purpose, the design focused on having the least amount of buttons and selectors possible to have a clean look that is not overwhelming for users.

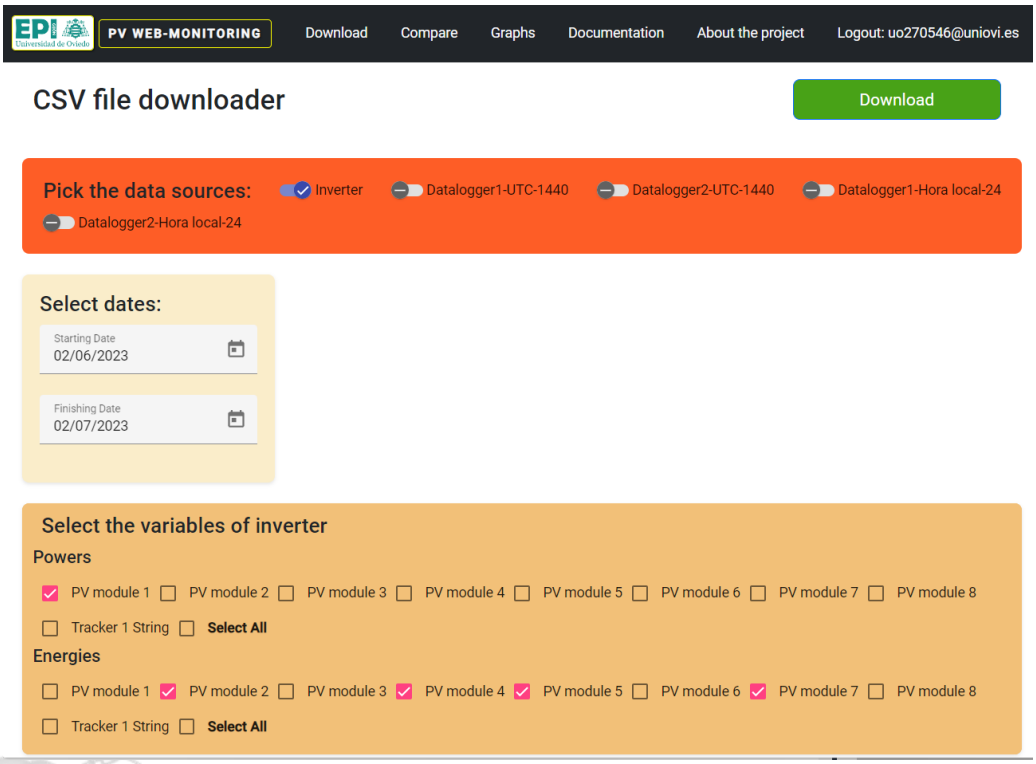


Figure 4.2.- Download page with inverter options open

The date pickers [27] bind themselves with two variables, startDate and finishDate which are initialized in the constructor to reference the month prior to the current date. These two variables also set the maximum and minimum date allowed for the opposite datepicker input, in this way, it can be assured that the time period is valid and the finishing date is after the starting date. These two variables are then used by all queries launched from the component using the **ConsultasService** functions. By doing this, if between two certain dates, you want to download more than one file to compare them, it would only require one click on the download button to do so.

Also following this pattern, there are only three filters for the columns, as there are only three sets of possible filters. This is because there are two pairs of sources who share magnitudes whose differences lie in the format, one has 1440 records (one per minute) and is on UTC time, while the other is on local time and has only one record per hour.

Code fragment 4.5.- download.component.html

```
<div class="row-title">
  <h2 class="title">CSV file downloader</h2>
  <button class="btn btn-primary btn-lg" (click)="submitForm()"
  [disabled]="!(inverter || datalogger1 || datalogger2 || datalogger24hl ||
      datalogger24hlestaci)">Download</button>
</div>

<section class="sources mat-checkbox-red">
  <h4>Pick the data sources:</h4>
  <mat-slide-toggle [(ngModel)]="inverter" color="primary">
  Inverter</mat-slide-toggle>
  <mat-slide-toggle [(ngModel)]="datalogger1"
      color="primary">Datalogger1-UTC-1440</mat-slide-toggle>
  <mat-slide-toggle [(ngModel)]="datalogger2"
      color="primary">Datalogger2-UTC-1440</mat-slide-toggle>
  <mat-slide-toggle [(ngModel)]="datalogger24hlestaci"
      color="primary">Datalogger1-Hora local-24</mat-slide-toggle>
  <mat-slide-toggle [(ngModel)]="datalogger24hl" color="primary">
  Datalogger2-Hora local-24</mat-slide-toggle>
</section>
```

```
<div class="date-picker-container">
  <h4>Select dates:</h4>
  <mat-form-field>
    <mat-label>Starting Date</mat-label>
    <input matInput [matDatepicker]="startPicker" [max]="finishDate"
        [(ngModel)]="startDate">
    <mat-datepicker-toggle matSuffix [for]="startPicker"></mat-datepicker-toggle>
    <mat-datepicker #startPicker></mat-datepicker>
  </mat-form-field>

  <mat-form-field>
    <mat-label>Finishing Date</mat-label>
    <input matInput [matDatepicker]="finishPicker" [min]="startDate"
        [(ngModel)]="finishDate">
    <mat-datepicker-toggle matSuffix [for]="finishPicker"></mat-datepicker-toggle>
    <mat-datepicker #finishPicker></mat-datepicker>
  </mat-form-field>
</div>

<div *ngIf="loading" class="spinner-overlay">
  <div class="spinner-border text-primary" role="status">
    <span class="visually-hidden">Loading...</span>
  </div>
</div>

<section class="selector mat-checkbox-yellow"
    *ngIf="datalogger1||datalogger24hlestaci">
  <h4>Select the variables Datalogger 1</h4>
  <mat-checkbox
      [(ngModel)]="filterData1.VM4T_E_Irrad_Global_Avg">VM4T_E_Irrad.Global_Avg
  </mat-checkbox>
  <mat-checkbox
      [(ngModel)]="filterData1.VM4T_E_Irrad_Difusa_Avg">VM4T_E_Irrad.Difusa_Avg
  </mat-checkbox>
    .
    .
```

```
  <mat-checkbox [(ngModel)]="selectAllDatalog1"
      (ngModelChange)="selectAllDatalog1Changed($event)"
  class="select-all-checkbox">Select All</mat-checkbox>
</section>
```

Therefore when the download button is pressed, many things can happen. If none of the checkboxes to select the data source is activated, the button will be disabled and the user will not be able to press it. And for every checkbox that is activated by the user, it will call one asynchronous function that utilizes the **ConsultasService** to make the query and download the incoming data in CSV format. This asynchronous function will take as parameters, the selected dates as well as the columns it wants.

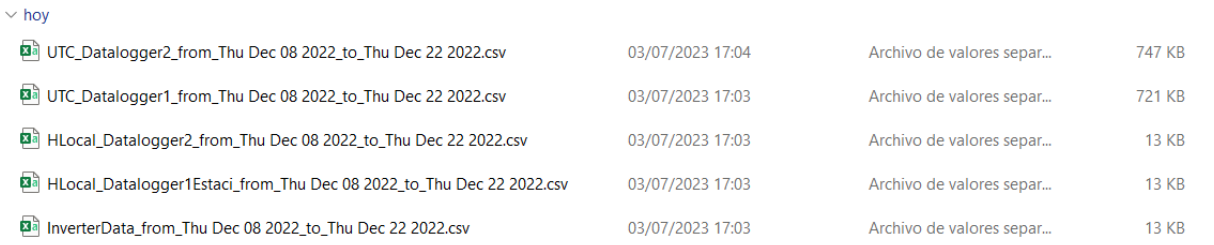


Figure 4.3.- Downloaded files

To decouple the filter from the variables needed, a new class with all possible magnitudes was created. These three new classes have one boolean member for each checkbox representing a column of the data source. The download component initializes these new objects by setting all of their members to **False**. Finally, these objects are passed as parameters to the functions of the corresponding service.

ConsultasService is the service in charge of fetching the Hypertext Transfer Protocol (HTTP) requests to the API. On **code fragment 4.6** the function in charge of getting the data from the first string shows the general schema that all functions on this service follow. It is noteworthy that only the parameters configured as True are included as query options. This approach aims to maintain simplicity in the query. The chosen HTTP method is GET, which serves the purpose of data retrieval without altering the server's state. This method is considered safe and idempotent.

Functions **convertToCSV** and **downloadFile** are auxiliary functions that turn the JavaScript Object Notation (JSON) received from the API into CSV format. As these functions may need to be used from anywhere in the application, it is a good practice to have them separated inside the ConsultasService.

Code fragment 4.6.- Function from ConsultasService

```
downloadInverterData(startDate: Date, endDate: Date, filters: InverterFilter): void {
    const apiUrl = API_URLS.InverterFilters; // API URL to fetch the JSON data
    let params = new HttpParams();
    params = params.append('StartDate', startDate.toDateString());
    params = params.append('EndDate', endDate.toDateString());
    // Append the selected variables to the request parameters
    for (const [variableName, variableValue] of Object.entries(filters)) {
      if (variableValue) {
        params = params.append(variableName, 'true');
      }
    }
    // Make the fetch request to get the JSON response
    this.http.get(apiUrl, { params: params }).subscribe((data: any) => {
      if (Array.isArray(data) && data.length === 0) {
        data = [{ ErrroMessage: 'NO DATA AVAILABLE AT THE PICKED DATES' }];
      }
      const csvData = this.convertToCSV(data); // Convert from JSON to CSV
      const fileName = `InverterData_from_${startDate.toDateString()}_to_
        ${endDate.toDateString()}.csv`;
      this.downloadFile(csvData, fileName); // Trigger the file download.
    });
}
```

4.1.3.- Graphical Information

While an in-depth analysis is performed with the raw numbers and a mathematical tool, a more direct approach can also be good to monitor a solar plant's state. The number of magnitudes measured and sensors can be overwhelming at times, and downloading a full year of all variables to rapidly check if all variables are being recorded properly turns

into a tedious task. For this and other objectives, it is important to have access to visual information.

Bearing this in mind and also the idea to compare with a simple look different dates, the **CompareComponent** and the **GraphsComponent** are designed. Both follow the same principle and utilize the same libraries and similar functions, however, the first is thought to analyze two different time periods side by side and the second one to examine a longer period of time to check trends and possible measurement errors.

The graphs tab, **figure 4.4**, shows a total of seven diagrams, displayed grouped by the magnitude they are measuring and the datalogger that registered the data. From datalogger1 the two different irradiations measured are being plotted, also the two temperatures it records. However, from datalogger2 there are, at the moment, up to nine different irradiations and five different temperatures. As well as those two magnitudes, the inclination of the string and the PM2.5 and PM10 particle density in the air.

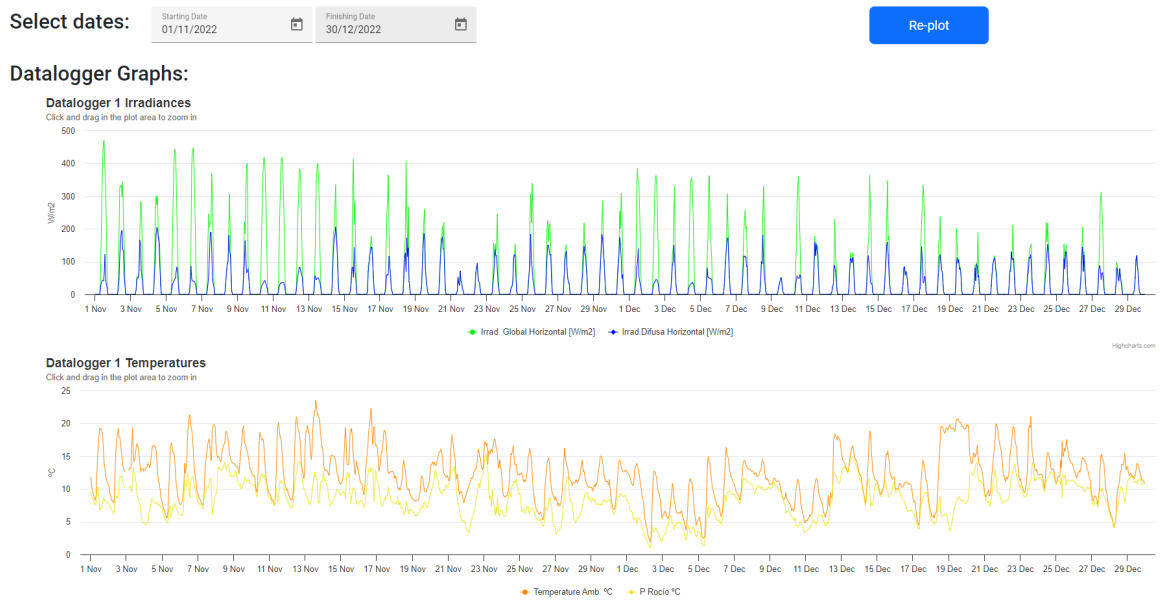


Figure 4.4.- Graphical information datalogger 1

As this tab is loaded on the full width of the tab, it is easier to look at larger date periods. Therefore, it is predefined to load the last month, but this can easily be modified by re-plotting all graphs on the screen or by zooming in on the exact graph you are

interested in. In order to zoom in, first you will need to click on the graph and drag to the desired period. If you want to zoom out to go back, a reset button is available to do so. To re-plot, you only need to use the date picker present at the top of the screen and click on re-plot. This will launch a new request to the API to get the data of that new period and load it onto the graphs.

The compare tab follows a very similar structure. It plots the same variables but splits the screen in half and has two different date pickers instead of just one. The pre-defined period is fifteen days for both, plotting on the right are the last fifteen days and on the left are the fifteen days prior to those on the right. In this tab, six queries are being executed instead of just three and are loading twice as many graphs too, but the total time period plotted is the same, one month.

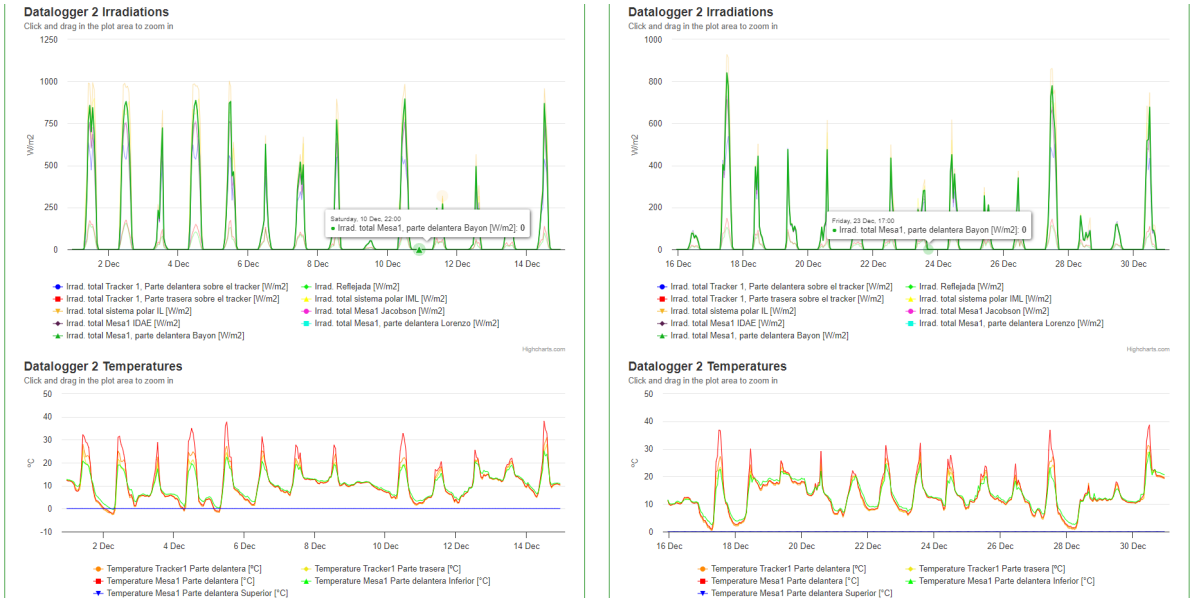


Figure 4.5.- Compare tab. Datalogger 2

Before loading the data onto each chart, it must be fetched from the API. In the case of the data from the inverter, it was simple to decide which information to plot and from where to obtain the data as there is only one data source for each string and only the energies are guaranteed to be present each hour. In the case of the two dataloggers, a decision had to be made regarding data representation. One option was to plot the data from the source with 1440 data points per day using the UTC timezone. Alternatively,

the data could be plotted using local time and represented by 24 data points per day. The choice was to simply use those on local time because of many reasons.

First and foremost because the inverter data was also in that format it was easier to compare quickly. Secondly, there were just too many values if it was decided to plot all the 1440 points at once therefore the time span must have been very small to obtain any conclusions by just looking at the diagrams. And finally, for performance reasons, as loading that amount of values took too much time and most of the points plotted were useless because at night the performance of the panels is irrelevant. For these reasons, it was settled to only 24 points per day and only those variables that are really relevant to keep track of over a long period of time.

Because of performance reasons, which will later be explained in **section 4.2** the queries done on these two tabs are done to a different route and only submit as query parameters the dates between which it wants the result. This speeds up the response from the Representational State Transfer (REST) API and therefore improves the loading time for both tabs. All of the data returned in JSON format is then parsed in pairs of values, the date and time of the measurement and the magnitude itself. These pairs are then passed as parameters to the functions in charge of initializing each graph. Those functions are called just after the pairs have been obtained and take one pair of data for each variable plotted on the graph. **Code fragment 4.7** shows the exact steps and configurations taken for loading the graphs.

Code fragment 4.7.- Chart initialization

```
getChartInverter(startDate: Date, endDate: Date): void {
    const apiUrl = API_URLS.Inverter_Graph; // API URL to fetch the JSON data
    let params = new HttpParams();
    params = params.append('StartDate', startDate.toDateString());
    params = params.append('EndDate', endDate.toDateString());
    // Make the fetch request to get the JSON response
    this.http.get(apiUrl, { params: params }).subscribe((data: any) => {
      if (Array.isArray(data) && data.length === 0) {
        data = [{ ErrorMessage: 'NO DATA AVAILABLE AT THE PICKED DATES' }];
      }
```

```
    let pair1 = data.map((entry: any) => {
            return [
              new Date(entry.Time).getTime(), // Convert date to milliseconds for
                  Highcharts
              parseFloat(entry['P1.1.1 E (Wh)'])];
            });
            **
            **
    this.initializeChartInverter(pair1, pair2, pair3, pair4, pair5, pair6, pair7,
        pair8, allString);

    initializeChartInverter(values: any[], values2: any[], values3: any[], values4:
        any[], values5: any[], values6: any[], values7: any[], values8: any[],
        valuesAll: any[]): void {
        this.chartInverter = {
          chart: {zoomType: 'x'},
          title: {text: 'Inveter data', align: 'left'},
          subtitle: {text: document.ontouchstart === undefined ?
              'Click and drag in the plot area to zoom in' : 'Pinch the chart to zoom
                  in', align: 'left'
          },
          xAxis: {type: 'datetime'},
          yAxis: {title: {text: 'Wh'}},
          legend: { enabled: true},
          **
          **
          series: [
          {name: 'E 1.1', data: values, color: '#FF0000'},
          {name: 'E 1.2', data: values2, color: '#00FF00'},
          **
          **
          {name: 'Srt 1.1', data: valuesAll, color: '#000000'}],
    };
}
```

In order to load the graphs onto the Document Object Model (DOM) it can be done with the **<highcharts-chart>** HTML tag after importing the highcharts module in the

desired component. **Code fragment 4.8** shows how this is performed. In **section 4.3** there is an explanation of why the decision to use the highcharts library for the graphs.

Code fragment 4.8.- HTML to load an inverter chart

```
<h2>Inverter Graphs:</h2>
<div class="inverter" style="display: flex; flex-direction: column;
justify-content: center; align-items: center;">
  <highcharts-chart *ngIf="chartInverter"
  [Highcharts]="Highcharts"
  [options]="chartInverter"
  style="width: 95%; height: 400px; display: block;">
  </highcharts-chart>
</div>
```

Even though in the two graphical tabs the periods are preset to one month of information, both of them can be modified to tailor to the need of the user. If the user desires to plot six months on either side, he can re-plot both of sides in the comparison component to fit that time period. That can be useful for very wide monitors, as the charts will stretch to fit the screen.

4.1.4.- Other tabs

This section discusses the significance of incorporating an **"About Us"** and a **"Documentation"** tab into the web interface, highlighting the benefits and functionalities they offer. They are static websites, unlike the rest of the web application and do not make any requests elsewhere to load information.

The "About Us" tab serves as a platform to introduce the project, providing users with a basic understanding of its purpose, scope, and stakeholders involved. Within this tab, users can access essential information about the project, including photographs of the solar panels deployed, visually depicting the scale and arrangement of the installation.

Furthermore, the "About Us" section includes a map showcasing the geographic location of the project. This adds a visual element to the interface and enables users

to identify the site where the monitoring system is implemented. In addition to visual content, the "About Us" tab includes contact information for the professors involved in the project.

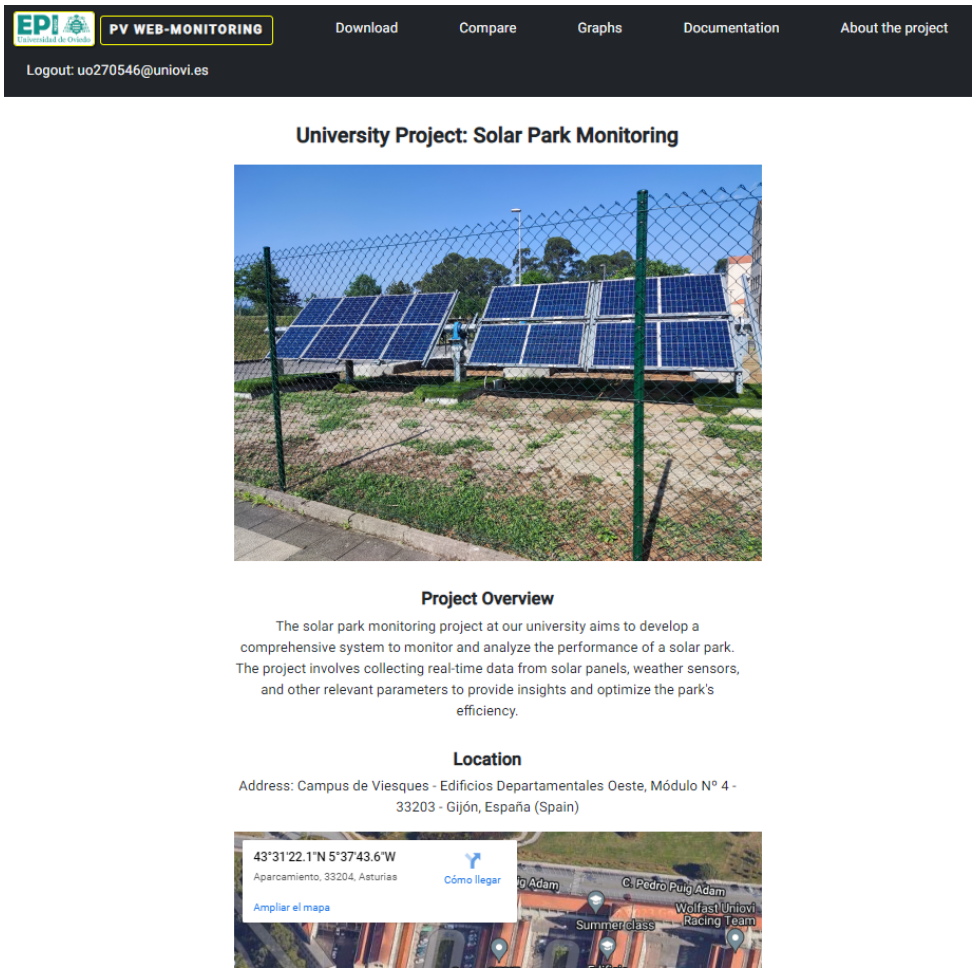


Figure 4.6.- About us window

The "Documentation" tab is also valuable for users seeking detailed technical information about the monitoring system. It serves as a repository of PDF documents that outline the monitoring system's components, architecture, and specifications. Furthermore, the documentation also includes photos as well as specific information regarding the solar panels themselves, such as datasheets, performance characteristics, and installation guidelines. This makes the documentation accessible from anywhere, and all centralized on the same site. The also page has images with links to the PDFs of the component shown and also tables with the key characteristics.

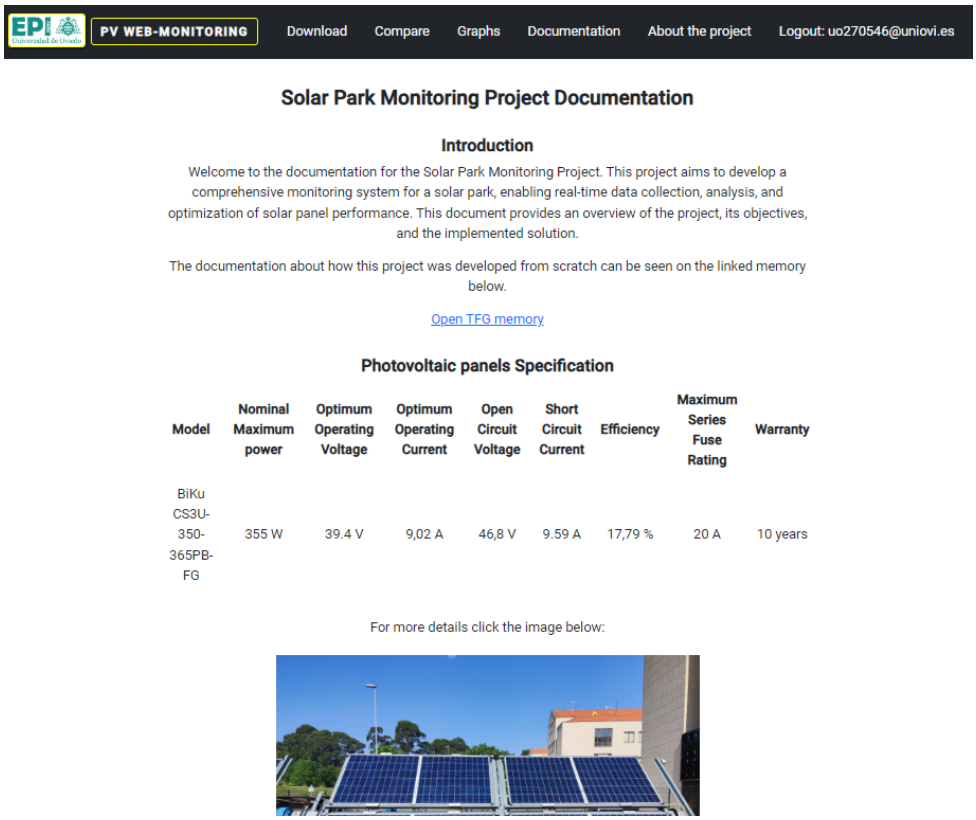


Figure 4.7.- Documentation window

This documentation also includes photos of the actual sensors placed on-site, as well as their brochures. As the sensors were already placed and configured before this project, their set-up and characteristics are not an important element of this project. Nonetheless, their inclusion on the site enhances the overall utility of the system by providing easy access to the most important documents related to the elements of the system.

By including an "About Us" and "Documentation" tab, the web monitoring interface improves user experience and enhances project comprehension.

4.2.- Backend

The backend of a web application refers to the server-side components that power its functionality and manage data processing. It encompasses the underlying infrastructure, databases, and application logic that work together to deliver the requested information or services to the user's browser. The backend is responsible for handling tasks such as data

storage, user authentication, server-side processing, and communication with external systems or APIs. Through the backend, developers can implement the business logic and handle the behind-the-scenes operations necessary for a web application to function smoothly.

In the case of this system, the client-side application needs to have access to the data stored on the repository CSV files filtered by date and magnitude. As it has been explained on **section 4.1**, the website creates an HTTP GET request with the dates and magnitudes it wants as query parameters. This fetch request is expecting a status code and a response in JSON format that it can parse to show the user.

For example, the GET request to receive all the variables on the inverters data source from the 1st of May to the 10th of June would be:

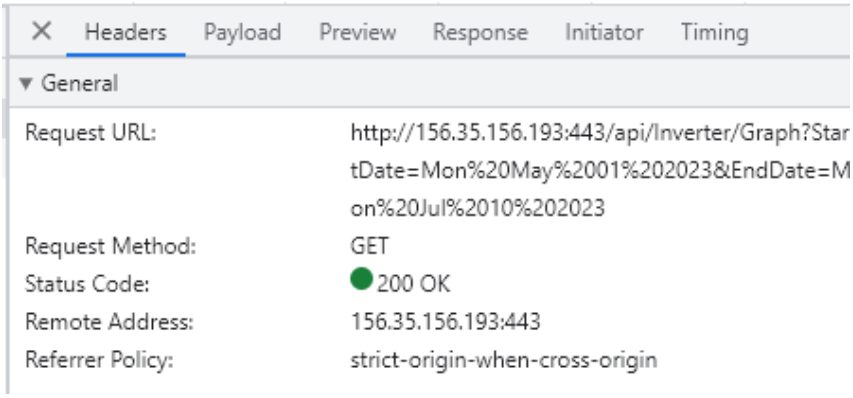


Figure 4.8.- Query to retrieve inverter data for plotting

4.2.1.- REST API

With this in mind, a REST API to serve this request started to be developed. REST is an architectural style, which emphasizes on a stateless, client-server communication model. REST is an architectural style which emphasizes a stateless, client-server communication model. Utilizing the IDE Visual Studio and ASP.NET core, a template for creating web API is already provided that scaffolds several characteristics needed to create the program. For example, it provides a swagger view to test the API, a default controller and the code needed to initialize it, as well as configuration for HTTP redirection.

The final design of the REST API consists of five different controllers, one per current data source, which serve all the HTTP requests related to that repository file. Inside each controller, there are two different functions, one that filters only by date, and another one which omits that process. By doing this, the graphical component would load faster as it won't need the extra computation done to filter out the columns that are not needed. Both functions return an OK response object with the JSON data response if no problems during the query occurred.

The process each function follows is almost identical, first, it opens a CSV file stored in the server where the code is run, and then it creates a dynamic list where it adds all rows that have a date between the two filters received as parameters on the query (**function 4.9**). Then it is serialized and further filtered by column if needed. Lastly, the resulting JSON is returned to the browser that made the request.

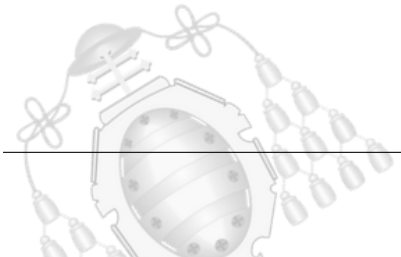
```
1        private List<dynamic> ReadCsvFile(Datalogger1Filters filters)
2        {
3            using var reader = new StreamReader(CsvFilePath);
4            using var csv = new CsvReader(reader, System.Globalization.
    CultureInfo.InvariantCulture);
5            var data = new List<dynamic>();
6            var records = csv.GetRecords<dynamic>();
7            foreach (var record in records)
8            {
9                if (IsWithinDateRange(record.Datetime, filters.StartDate,
    filters.EndDate))
10               {
11                   data.Add(record);
12               }
13           }
14           return data;
15       }
```

Code fragment 4.9.- ReadCsvFile function

The filter coming from the query parameters is parsed into an object model defined in a separate file. In this way, it can be modified easily and the code can be more flexible to add newer filters related to newer variables present on the dataloggers or inverter files. There are three models, all sharing the **startDate** and **endDate** boolean values. These models reference each of the magnitudes measured at the moment by the system. All file paths are also defined in a separate folder called **FilePaths.cs** to keep insisting on the modular aspect of the system. In this way when new sources are added, the process to create a new controller to serve those requests will be almost trivial.

The code provided on **4.10** contains the definition of the controller that handles HTTP GET requests made about the data coming from the first string. Here's an explanation of what the code does:

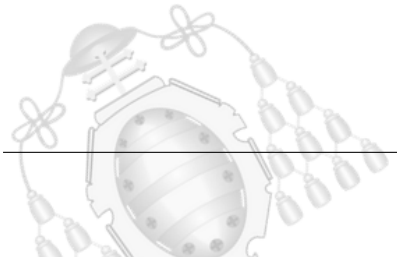
- The controller is defined with the [**ApiController**] and [**Route("api/[controller]")**] attributes, indicating that it is an API controller and that its routes will start with "api/inverter" as "inverter" is the name of the controller itself.
- The "CsvFilePath" constant is defined, which represents the file path of the CSV file containing inverter data inside the server. It is obtained from the class **FilesPaths**.
- The **Get** function is an HTTP GET action that takes an optional query parameter of type"'InverterFilter". This method retrieves and filters inverter data based on the filters provided.
- **ReadCsvFile** is the function called afterwards with the provided date filters. This function reads the CSV file specified by "CsvFilePath" using an external library called **CsvHelper**.
- The retrieved data is then serialized to JSON format using "JsonConvert.SerializeObject" method and assigned to the "jsonData" variable.
- The 'FilterJsonData' function is called with the "jsonData" and the provided filters. This function applies the filters to the full JSON data and returns it filtered.
- Finally, the selected JSON data is returned as the HTTP response.



```
namespace APIsolarMonitoring.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    public class InverterController : ControllerBase
    {
        private const string CsvFilePath = FilesPaths.InverterString1;

        [HttpGet]
        public ActionResult<IEnumerable<dynamic>> Get([FromQuery]
    InverterFilter filters)
        {
            var data = ReadCsvFile(filters);
            var jsonData = JsonConvert.SerializeObject(data);
            var filteredJson = FilterJsonData(jsonData, filters);
            return Ok(filteredJson);
        }
        *
        *
```

Code fragment 4.10.- InverterController.cs

The other four controllers operate in a similar way and only differ on the specific functions in charge of filtering the data by column. For the queries done to fill the graphs, like the one on **4.8**, the routed function skips the third line of the **ActionResult** function and returns the data as it comes after serialization. The final architecture of the API is shown on **figure 4.9**. Note that only those sources that are being plotted have two different endpoints.



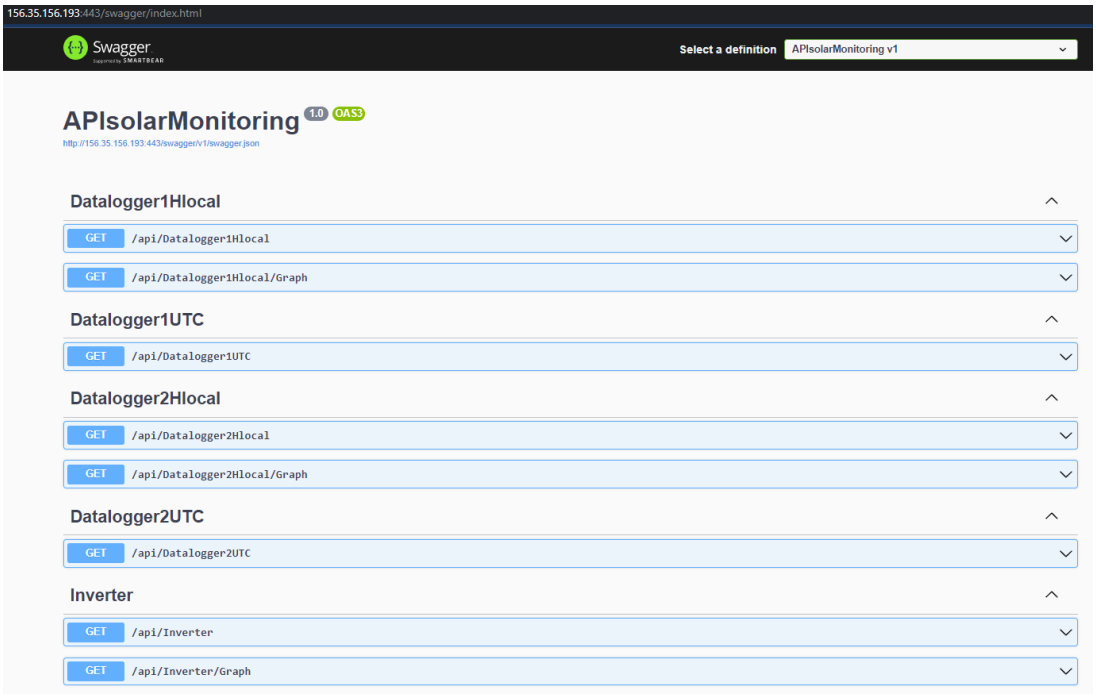


Figure 4.9.- Swagger documentation

4.2.2.- Cloud-based authentication using Firebase

For the user authentication, the possibility to develop it from scratch using the .NET framework was quickly discarded due to time constraints and security concerns. In fact, it could be a whole project by itself. Instead, the idea to use a cloud-based API to use the authentication seemed simpler as well as more robust.

Firebase is an excellent choice for web application authentication due to its affordability and seamless integration with Angular. With Firebase, developers can implement secure and user-friendly authentication without incurring extra costs or complexities. When integrating Firebase authentication into an Angular project, the process is straightforward and well-documented [24]. Angular, being a widely used JavaScript framework, benefits from Google's association with both Angular and Firebase. This close relationship ensures smooth integration, clear guidelines, and ample support for developers. By following the provided documentation and guidelines, setting-up Firebase authentication within the Angular project was made very easy and straightforward.

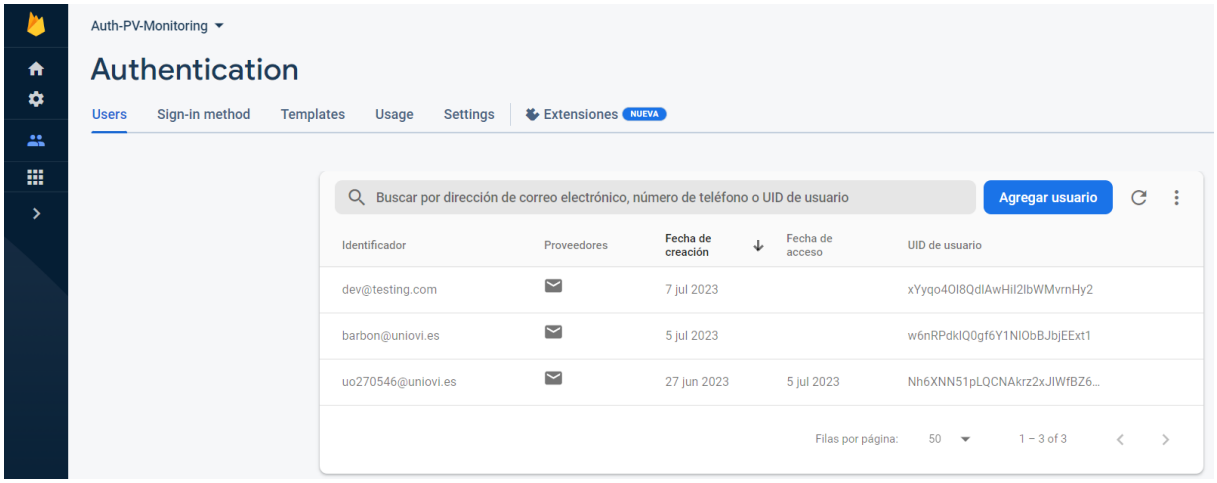


Figure 4.10.- Firebase configuration window

To create a Firebase project, first, a Google account must be provided, this account will have the role of network administrator as it will have the ability to create and delete users freely. For this project creation, the process is guided throughout the Graphical User Interface (GUI) and its setup is trivial. Then inside the project, after selecting **Authentication**, a menu appears where how the users can authenticate to your application has to be decided. In this case, an email and a password will be the only possibility, however, it can later be modified. Inside the process of user creation is very simple.

4.3.- Frameworks chosen

While the goals of a project may be somewhat clear, the path to achieving these goals is almost certainly diffuse at first glance. Many decisions have to be taken, resulting in different challenges during development and providing a different quality product at the end. Nowadays, web development is in a very mature state, and there are many options to choose from regarding frameworks and even languages. Each project is unique and therefore the decisions taken for a different project may differ significantly from one another, depending on factors such as project requirements, scalability, maintainability, and developer expertise. In this section, an exploration will be conducted into the

particular decisions made throughout the development process, emphasizing the rationale behind these choices and their influence on the ultimate outcome of the product.

4.3.1.- Angular

Figure 4.11.- Angular's logo

Angular is a powerful and modern framework that provides a modern and interactive user experience. It is a client-side framework that allows us to create dynamic and responsive web applications. Some of the reasons why Angular was chosen for the frontend development of our solar park monitoring website:

- Robustness and Scalability: Angular is designed to handle complex and large-scale applications, this is why it is very popular, especially among large companies. It also provides a solid foundation for building robust and scalable web applications, offering features like dependency injection, modular architecture, and comprehensive tooling that enhance maintainability and scalability.
- Two-way Data Binding: Angular's two-way data binding enables automatic synchronization of data between the application's model and the view. This feature simplifies development by reducing the need for manual DOM manipulation and ensures consistent and up-to-date data representation.
- TypeScript-based: Angular is written in TypeScript, a superset of JavaScript that provides static typing and other features that make it easier to develop and maintain. TypeScript helps to catch errors at compile-time rather than at runtime, which makes the code more reliable and easier to debug.
- Component-based Architecture: Angular follows a component-based architecture, where applications are built as a collection of reusable and modular components.

This approach promotes code reusability, separation of possible problems, and easier collaboration among developers, resulting in more maintainable and extensible codebases.

- Powerful Templating and Directives: Angular offers a powerful templating system that allows developers to create dynamic and interactive user interfaces declaratively. Additionally, Angular's extensive set of built-in directives enables the creation of custom behaviours and interactions, providing flexibility and enhanced functionality.

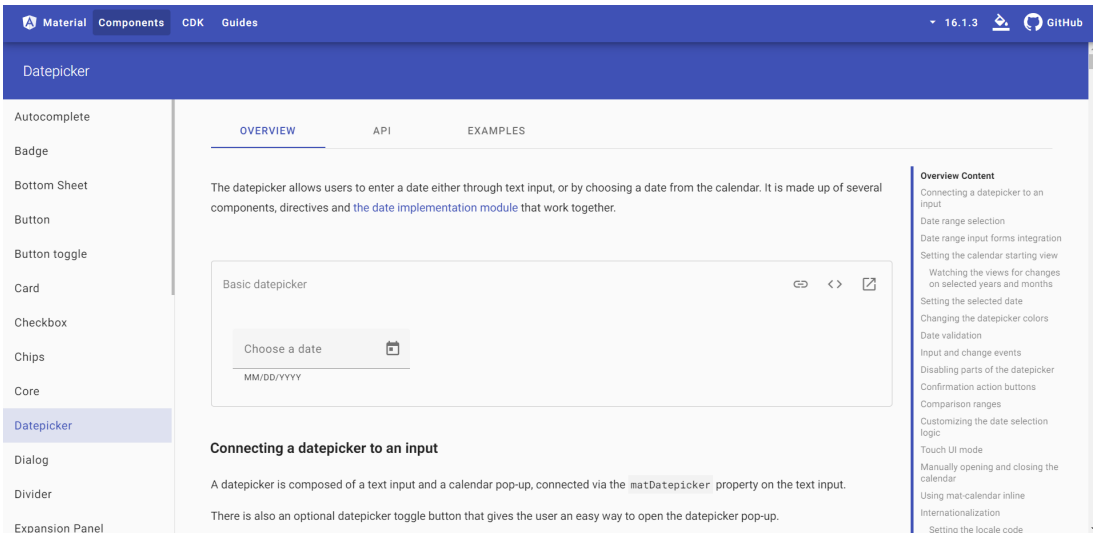


Figure 4.12.- Angular material documentation [30]

Some notable alternatives were the ASP.NET core framework, as it would have made the integration with the API very seamless, and also because it is very simple and well documented. Or React, as it is the most popular JavaScript framework right now and would make a very useful and on-demand skill to acquire.

4.3.2.- ASP.NET Core

For the REST API, Microsoft's .NET Core framework was the elected choice. This framework is very popular among developers and especially companies, thanks to the incredible environment it has, the support and the number of functionalities.

.NET Core is an open-source, cross-platform framework developed by Microsoft. It provides a runtime environment and a set of libraries that enable developers to build various types of applications. .NET Core supports multiple programming languages such as C#, Visual Basic, and F#, and it offers a range of features and functionalities for building efficient and scalable applications. ASP.NET Core, on the other hand, is a web framework built on top of .NET Core. It is specifically designed for creating web applications and APIs.

Some reasons that were taken into account when making this decision were:

- Cross-platform Compatibility: ASP.NET Core is designed to be cross-platform, enabling the development of web APIs that can be deployed and run on different operating systems, such as Windows, Linux, and macOS. This flexibility allows for broader reach and compatibility across various hosting environments.
- Security and Authentication: ASP.NET Core provides robust security features and built-in authentication mechanisms, allowing for secure communication and data protection. It supports industry-standard security protocols and encryption algorithms, ensuring the confidentiality and integrity of data exchanged with the API.
- Developer Productivity and Tooling: ASP.NET Core offers a rich development ecosystem, providing a wide range of tools, libraries, and frameworks that enhance developer productivity. It integrates very well with popular development tools, such as Visual Studio, offering seamless development experiences and extensive debugging capabilities.
- Templates: One advantage of using ASP.NET Core for web API development is its powerful templating system, which simplifies and accelerates development. ASP.NET Core provides a variety of templates and scaffolding options that offer preconfigured project structures and ready-to-use code snippets. These templates save developers time and effort by automating common setup tasks, allowing them to focus on implementing business logic and unique application features. Additionally, the scaffolding feature generates code for Create, Read, Update, Delete (CRUD) operations based on specified data models, streamlining the implementation of API

endpoints and data access. Overall, ASP.NET Core's templating system enhances developer productivity and expedites the development process.

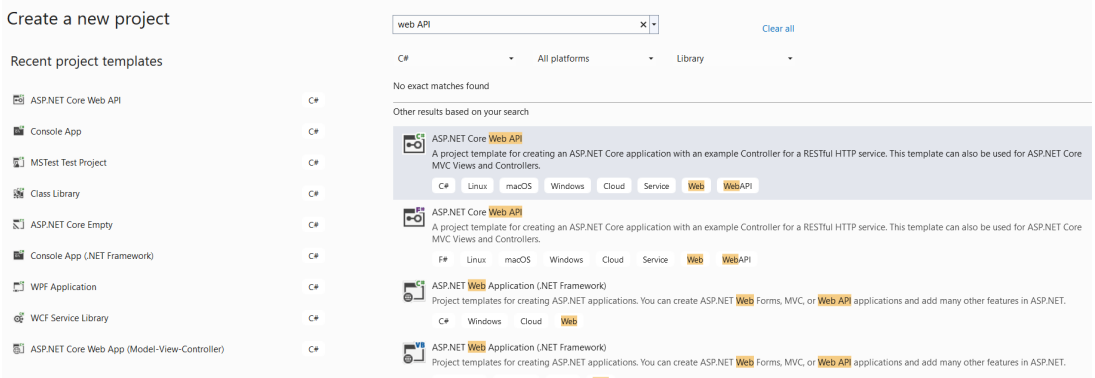


Figure 4.13.- Template for a Web API

4.3.3.- Highcharts

When deciding on a charting library for this project, there are many important aspects to consider: integration ease, versatility, and visual customization options.

Highcharts, specifically, offers a diverse selection of chart types, including line charts, bar charts, and pie charts, among others. This broad range covers the most frequently encountered data visualization requirements, enabling effective presentation of data in an appealing and comprehensible manner. Moreover, Highcharts enhances the user experience through interactive features like zooming, and panning. These functionalities contribute to a more engaging and dynamic interface.

Comparing Highcharts to Chart.js, both are popular charting libraries, but Highcharts offers a richer set of diagrams and more customization. It also gets a decisive advantage in the integration with the JavaScript framework of choice for this project, Angular, as the website of the library [28] provides a step-by-step process to set-up a demo chart on an Angular project. While Chart.js is lightweight and easy to use, Highcharts provides more extensive documentation, support, and a larger community, making it easier to find solutions to potential issues and access additional resources, like the ones on **4.14**.

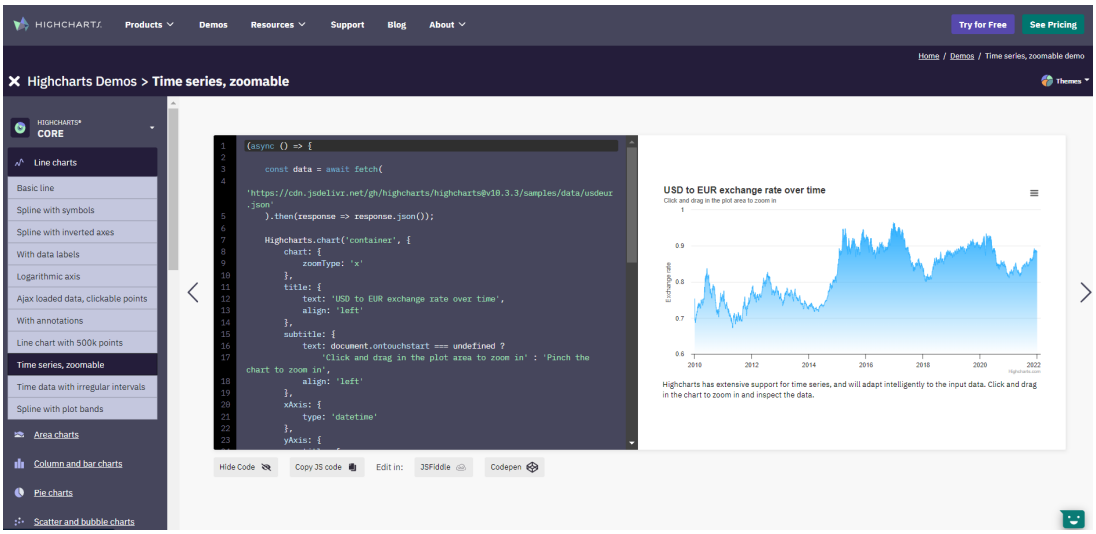


Figure 4.14.- Documentation integration highcharts [29]

Moreover, Highcharts provides extensive customization options, allowing the developer to tailor the charts to its specific project requirements. The possibility to customize colors, fonts, labels, and other visual elements, ensures that the charts align with this project's design. Highcharts also supports responsive design, enabling the charts to adapt and scale appropriately across different screen sizes and devices. This is especially useful for the compare component that allocates only half the screen to each diagram and has a lot of values to plot.

Considering the educational nature of this project and the availability of the free version of Highcharts, it becomes an even more compelling choice. Highcharts' free version still offers a wide array of capabilities, making it suitable for almost every project, the only downside is the small watermark on the side of the charts with a link to its website.

In summary, Highcharts is a great choice for an Angular project due to its rich feature set, extensive customization options, and strong community support. It offers an excellent balance between ease of use and advanced functionality, making it well-suited for a web project of these characteristics.

5. Deployment

For the deployment of the system, a computer running on Windows 10 was used as the server and focal point of the monitoring service. This computer is located in the West Department building in the attic of the fourth module. This decision was a mixture of the requirements of the project and a decision on which operating system to use. The deployment process included the creation of the file organization system shown in 3.9, the deployment of the website and the REST API on a web server, as well as the download of all tools and programs needed to get and modify the code.

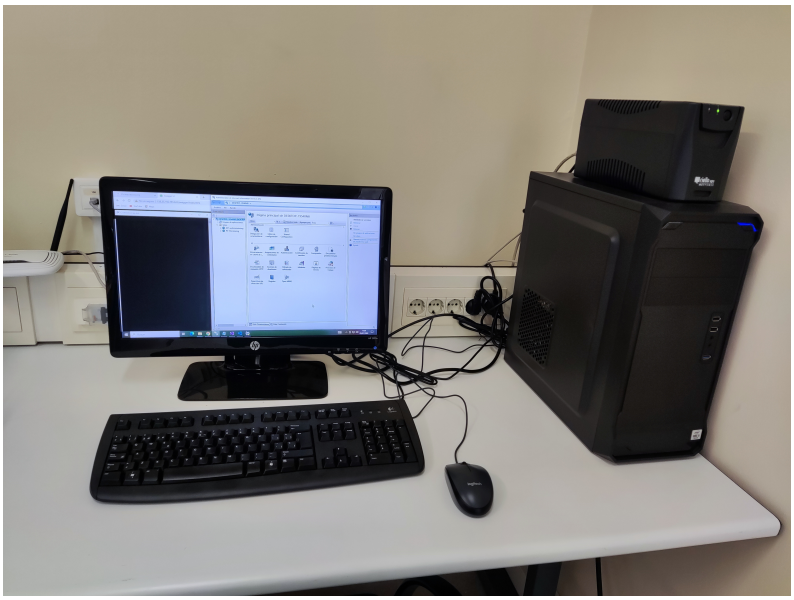


Figure 5.1.- Computer used as server

The first step was to create all the necessary folders on the computer to store the files needed. As the computer was formatted beforehand, there were no unnecessary files. The folders were created following a tree structure inside the **Documents** folder. Once all the files and folders had been organized, the specific programs were installed. First, the latest version of Python was installed from the official website [31]. Then with Python installed, from the terminal and with the command **py -m ensurepip –upgrade** pip, python's package installer was installed. Now, the libraries needed for the scripts could

get installed. Command: **py -m pip install pandas** installs the pandas library, and just substituting "pandas" by the name of the library which wished to be installed, all other libraries were set up. Next, git was installed on the machine to be able to pull the code from the repositories where it was hosted. It was a key element in this phase as it made it very easy to have all the code accessible on the local machine. Then, both **Visual Studio** and **Visual Studio Code** were installed to be able to edit the code and compile it for production.

Afterwards, in order to be able to build the frontend, Angular was installed. To do so first you **node.js** must be installed first, and only then AngularCLI can be installed using node package manager (npm) with the command: **npm install -g @angular/cli**.

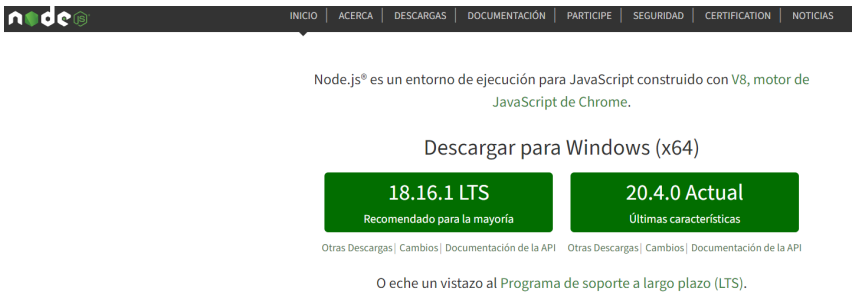


Figure 5.2.- Nodejs installation page

Finally, the programs could get pulled from the repositories and compiled for deployment. Using Visual Studio Code's source control, to have the code locally the desired repository must be cloned. This is done by copying the link of the repository into the input that appears after clicking on **Clone repository**. This process is repeated three times on different folders, twice for the python scripts and once for the whole frontend project. Then, inside the folder where the Angular project is stored, a last installation must be performed using npm, executing **npm install** on the main folder of the project, Node will install all required packages listed on the package.json file. Lastly, from Visual Studio Code, the IDE used to develop the web API a similar process was performed to clone the remote repository onto the machine.

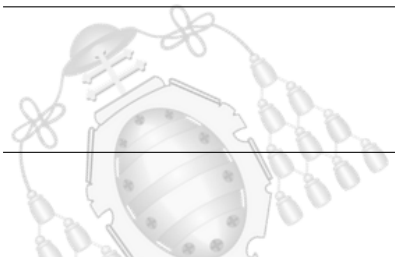
Upon completing these steps, and following the manual modification of all routes directing to files and folders, ensuring their compatibility with the server environment

rather than the development machine, the data processing scripts were executed. To verify the proper functioning of all components, extensive testing was conducted using additional real data. Additionally, both sections of the website underwent rigorous testing on a local development server to confirm smooth operation across all aspects.

Both the website and the API were deployed on the computer utilizing **Internet Information Services (IIS)**. This software comes pre-installed in all Windows 10 machines but must be activated first. To do so navigate to Control Panel > Programs > Programs and Features > Turn Windows features on or off. Then select the Internet Information Services checkbox and click on OK. Once this is done, some extensions to the service needed to be installed for everything to work well, those are: URL re-write [32] and ASPNetCoreModulev2 [33]. The first module is used to configure the routes to make them compatible with the routing used by Angular. Its contents are manually set and are shown below.

Code fragment 5.1.- web.config

```
<?xml version="1.0" encoding="utf-8"?>
    <configuration>
      <system.webServer>
        <rewrite>
          <rules>
            <rule name="Angular Routes" stopProcessing="true">
              <match url=".*" />
              <conditions logicalGrouping="MatchAll">
                <add input="{REQUEST_FILENAME}" matchType="IsFile" negate="true" />
                <add input="{REQUEST_FILENAME}" matchType="IsDirectory" negate="true" />
              </conditions>
              <action type="Rewrite" url="/" />
            </rule>
          </rules>
        </rewrite>
      </system.webServer>
    </configuration>
```



Firstly, to deploy the frontend on the web server, a fixed IP must be assigned to the computer. This is key because the way to access the website will be by writing the IP on the browser as it does not have a domain name. This process was performed by a teacher of the university with permission to do so. To deploy the website on the server, first, the angular project must be built for production, the command to do so is: **ng build –production –aot**. After this, the contents from the dist folder inside the angular project must be copied onto the desired directory. This is the directory that must be referred by the new website created on IIS manager GUI. Finally, a port and an IP address must be selected to host this website. The port chosen for the frontend is port 80.

To deploy the REST API the process is almost identical. To build the application for production, in this case, from the GUI of Visual Studio, it is possible to publish the binaries onto any chosen folder. Then the same process described above is followed but this time selecting port 443. Both applications are therefore on the same IP but utilize different ports.

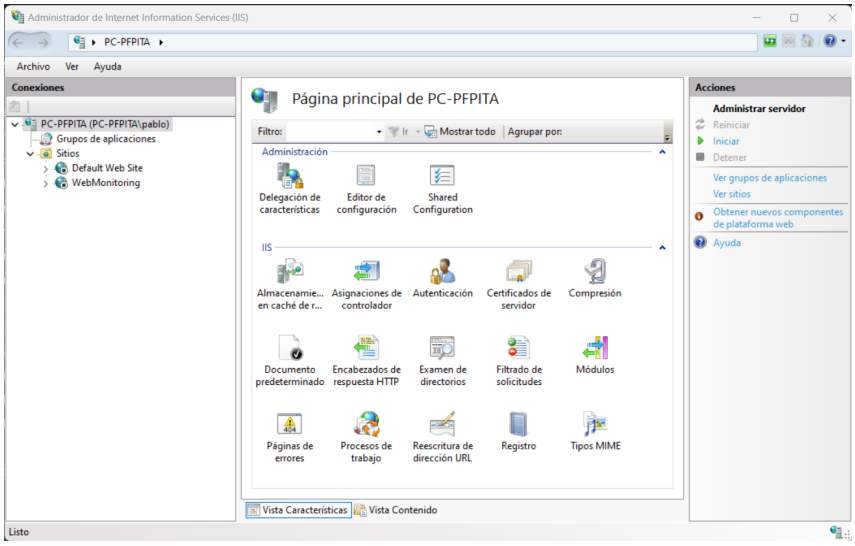


Figure 5.3.- IIS Manager interface

Deploying a website on IIS using port 80 and an **API** on port 443 serves distinct purposes and aligns with specific security considerations. Port 80 is commonly used for unencrypted HTTP traffic, making it suitable for hosting the website. By utilizing port 80, the website can be accessed by users without the need for additional encryption protocols.

This setup simplifies the browsing experience for visitors, as they can directly access the website using standard HTTP requests.

On the other hand, port 443 is typically reserved for encrypted Hypertext Transfer Protocol Secure (HTTPS) traffic. Deploying the API on this port ensures secure communication between clients and the server by leveraging the Transport Layer Security (TLS) or Secure Socket Layer (SSL) protocols. HTTPS encrypts the data exchanged between the REST API and its consumers, providing confidentiality and integrity of sensitive information. Utilizing port 443 for the API ensures that data transmitted through the API is protected against unauthorized access, eavesdropping, and tampering.

Figure 5.4.- Firewall rules

By segregating the website and API onto separate ports, port 80 and port 443 respectively, it allows for efficient management and control of traffic flow, while simultaneously ensuring the appropriate security measures are implemented for each component.

As has been mentioned before in this chapter, the computer is running Windows 10 and it is hosting two web services utilizing the IIS because of a number of reasons, not all of them related to all-around performance or possibilities. The decision to do so is motivated by the following factors:

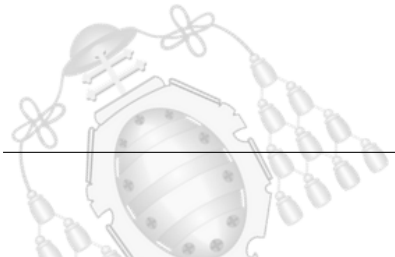
- Familiarity and ease of use: Windows 10 is a widely used operating system, and many non-technical users are already familiar with its interface and functionalities. This familiarity can make it easier for teachers or other students to navigate and manage the server environment as well as the files stored.
- Compatibility: IIS is a web server software developed by Microsoft specifically for Windows operating systems. Its integration simplifies the setup process and reduces complexity.

- Availability: the computer was already provided with Windows 10 installed and for the requirements of the project it was simpler to just use the default web server it comes with. It would be a waste of resources to use a cloud provider to host the website when it can be manually done on the computer itself.
- Support and documentation: Windows operating systems, including Windows 10, have a vast support network and a wide range of documentation available. In case in the future any issues or questions appear, resources, tutorials, and forums to seek assistance are easily found.
- Robust security features: it provides a range of security features that can help protect the server and the deployed website. The integrated security features can be configured through a GUI, allowing teachers or other users to easily set up and manage security settings, SSL certificates, and access controls.

Other possible alternatives included using Ubuntu's distribution of Linux, using Apache web server or even deploying the app on a known cloud provider like AWS.

The links to both the website and the online documentation to test the web API are:

- Web monitoring service: http://156.35.156.193/
- Swagger documentation of the REST API: http://156.35.156.193:443/swagger/index.html



6. Planning

A key aspect of any engineering project is the planning and budgeting of the project itself. In big companies, projects have many roles only assigned to these two tasks. However in this case, as it is a solo project, all roles are centered around the same person. This complicates efficient time management and human resource planning but it does not make it impossible.

The presented table outlines the project's budget, offering a comprehensive view of the estimated hours and corresponding costs associated with the involved tasks. By dividing the project into three distinct jobs - data cleaning and storage, web development, and server deployment - specific tasks are delineated for professionals with varying expertise. This breakdown of the budget serves as a point of reference for estimating costs, considering the requisite capabilities required to accomplish each task.

The time writing the report itself is included in each of the roles, representing an actual report of the job just done. This way the table is simpler and cleaner. The total amount of hours is estimated using the planning hours and calculating an average of 8 hours of work a day during the last month and a half.

As all software components used were open source or using the trial version, no extra cost is associated with the project. Physical devices already owned by the university will not be taken into account either, as they were already there, but with a different or less important role. This includes the solar panels, pyranometers, weather station, particle sensor, dataloggers, the computer used as a server, the cables used to connect to the sensors, electricity, the inverter and the optimizers.

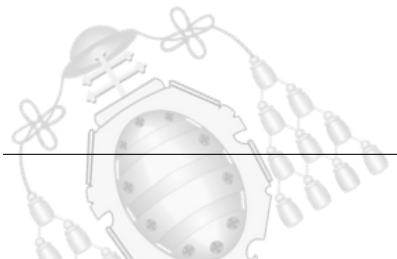
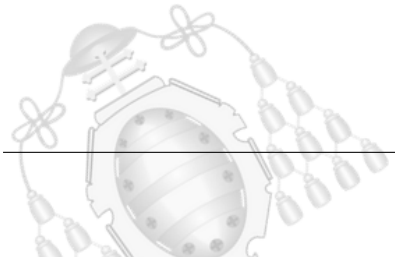


Table 6.1.- Project Budget

Role	**Hours**	**Cost per hour**	**Total cost**
Data scientist	80	35 €/h	2.800 €
Full-stack web-developer	240	30 €/h	7.200 €
Deployer	6	20 €/h	120 €
Total	**350**		**10.120 €**

The planning followed to develop the whole system is detailed in the Gantt chart below. It includes the core parts of the process and their time span. Not all days were equal and the time dedicated to each task is not proportional to the days allocated to it. In the last two months of the project, a substantial increase in working hours was performed in order to achieve the deadlines set for the whole project.

The software used to design this strategy was Microsoft Project 2013.



	Task Name	Duration	Start	Finish	Predecessors	Reso
0	**Web monitoring system**	**115 days**	**Mon 06/02/23**	**Fri 14/07/23**		
1	Documentation and requirements analysis	41 days	Mon 06/02/23	Sun 02/04/23		
2	Dataloggers processing script	55 days	Thu 09/02/23	Thu 27/04/23		
3	Frontend development	49 days	Thu 27/04/23	Wed 05/07/23	2	
4	Inverter script	14.5 days	Tue 16/05/23	Mon 05/06/23	2	
5	REST API	21 days	Fri 02/06/23	Fri 30/06/23		
6	Authentication	5 days	Wed 28/06/23	Tue 04/07/23		
7	Deployment and testing	9 days	Sun 25/06/23	Wed 05/07/23		
8	Budget	4 days	Thu 06/07/23	Tue 11/07/23	7	
9	Post Implementation Review	3 days	Wed 12/07/23	Fri 14/07/23	8	

Figure 6.1.- Gantt chart of planification

7. Conclusions and future improvements

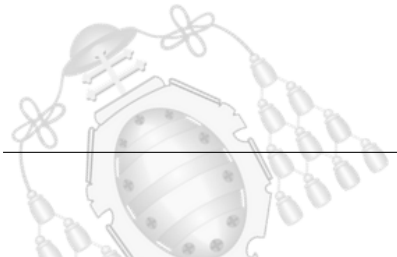
Throughout the duration of the development process, numerous challenges were encountered, providing valuable opportunities for personal growth and professional development. Despite moments of uncertainty and daunting tasks that initially appeared overwhelming, the remarkable results achieved serve as a testament to the perseverance and determination exhibited. The journey proved to be an enlightening experience, enabling the acquisition of essential skills to operate independently and efficiently, with minimal reliance on external deadlines, ultimately working towards the fulfilment of the objective.

From the moment the project was started to the time it is presented, almost seven months have gone by with a single objective in mind, to deliver a good quality system that fulfils all the requisites and meets the expectations set at the start.

The goal of this report is to showcase the intentions behind every feature and give a general idea of how most things work, as well as how much time and effort has gone into it.

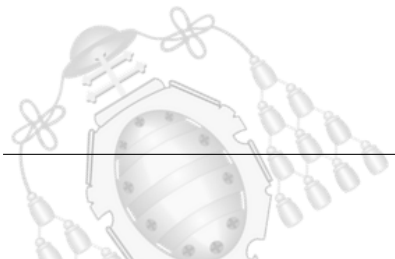
During the whole development of this project, an immense amount of knowledge has been acquired; as developing and maintaining this web monitoring system has been a great addition for this last part of the degree.

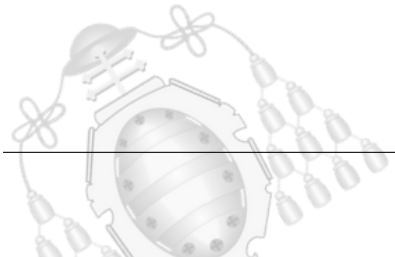
Although it is not perfect and could be polished better, the results are noteworthy and make for a very solid project. Because of time and skill constraints, there are many possible upgrades to the system that could ease out the work of the teachers working on this analysis or at least improve the general performance. Some possible improvements that could get added to the system are:



- Add a new script to process the data coming from Arganda del Rey: following a very similar approach to the one followed for the dataloggers installed on the campus in Gijón but tailored to the specifics of that data.
- Adding the second string as a new source on the website: to allow all existing functionality on this new source.
- Visual enhancements: improve the overall appearance of the website itself. Centring some buttons, modifying some input sections, or adding a footer are some of the possible additions to have a more modern and polished site.
- Creating newer graphs: with magnitudes from different sources that are relevant to have together on the same diagram.
- Improving the script in charge of processing the inverter's data: to allow for one-day files or multiple-day files to be added at once to the system and consequently reduce the time spent downloading this raw data on SolarEdge's site.
- Buying newer dataloggers: with more capabilities such as sending the CSV file generated on the day directly to the computer acting as server without the need to access manually to the SD card.

Overall, the project has successfully achieved all its initial goals and even additional objectives that emerged during the development process. The end product is a source of pride, particularly due to its potential to support the exceptional teachers who provided their guidance throughout this project in their research on the intricate aspects of solar energy generation. It is hoped that the project can effectively facilitate their research efforts and contribute to the production of compelling papers that advance the field of renewable energies.





Bibliography

[1] SolarEdge Software https://www.solaredge.com/en/products/software-tools

[2] https://www.fronius.com/es-es/spain/energia-solar/instaladores-y-socios/productos-y-soluciones/herramientas-digitales-y-de-monitorizacion-para-tu-sistema-fotovoltaico/sistema-fv-monitorizacion-solarweb

[3] https://www.sma-iberica.com/

[4] https://www.solarwholesalers.com.au/solar-web-monitoring

[5] https://www.sma.de/es/productos/monitorizacion-y-control/webconnect

[6] Shaheer Ansari, Afida Ayob, *A Review of Monitoring Technologies for Solar PV Systems Using Data Processing Modules and Transmission Protocols: Progress, Challenges and Prospects* https://www.researchgate.net/publication/353355477_A_Review_of_Monitoring_Technologies_for_Solar_PV_Systems_Using_Data_Processing_Modules_and_Transmission_Protocols_Progress_Challenges_and_Prospects

[7] https://tuvatio.es/blog/como-funcionan-paneles-solares/

[8] https://shorturl.at/CEHOY

[9] Hussein A Kazem, Miqdam T Chaichan *Effect of Humidity on Photovoltaic Performance Based on Experimental Study* https://www.researchgate.net/publication/289546072_Effect_of_humidity_on_photovoltaic_performance_based_on_experimental_study

[10] https://www.kippzonen.com/Product/416/LOGBOX-SE-Data-Logger

[11] https://www.sunslewdrive.com/

[12] https://www.python.org/

[13] https://www.jetbrains.com/es-es/pycharm/

[14] https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html

[15] https://jupyter.org/

[16] https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.interpolate.html

[17] Python Pandas Tutorial: Cleaning Data - Casting Datatypes and Handling Missing Values https://www.youtube.com/watch?v=KdmPHEnPJPs&list=RDCMUCCezIgC97PvUuR4_gbFUs5g&start_radio=1&rv=KdmPHEnPJPs&t=11&ab_channel=CoreySchafer

[18] https://www.geeksforgeeks.org/how-to-remove-timezone-from-a-timestamp-column-in-a-pandas-dataframe/

[19] Programming languages used in the field of data science https://shorturl.at/cpzEU

[20] https://pandas.pydata.org/docs/reference/api/pandas.readcsv.html

[21] https://pypi.org/project/suntime/

[22] https://www.pola.rs/

[23] https://medium.com/cuenex/pandas-2-0-vs-polars-the-ultimate-battle-a378eb75d6d1

[24] https://github.com/angular/angularfire/tree/master/site/src/auth

[25] https://www.youtube.com/watch?v=8VTxuIvMTlc&ab_channel=Garajedeideas

[26] https://material.angular.io/components/snack-bar/overview

[27] https://material.angular.io/components/datepicker/overview

[28] https://www.highcharts.com/blog/integrations/angular/

[29] https://www.highcharts.com/demo

[30] https://material.angular.io/components/categories

[31] https://www.python.org/downloads/

[32] https://www.iis.net/downloads/microsoft/url-rewrite

[33] https://dotnet.microsoft.com/en-us/download/dotnet/7.0

[34] https://learn.microsoft.com/en-us/aspnet/core/tutorials/publish-to-iis?view=aspnetcore-7.0

[35] https://levelup.gitconnected.com/how-to-deploy-angular-app-to-an-iis-web-server-complete-setup-337997486423

[36] https://es.stackoverflow.com/questions/262137/ng-build-prod

[37] https://www.youtube.com/watch?v=5m_R4vxjTkk&ab_channel=render2web

A. Appendix: Code Repositories

All the code written to complete the project can be accessed from the links below. There are in total four repositories, two for the data processing and two for the website. As explained in **chapter 6** the first scripts developed were those related to the data pipeline, one for the dataloggers and one for the inverter. The frontend and API also have one repository each that was key during the deployment phase.

- Dataloggers script: https://github.com/GaremoP/CSVdataParser
- Inverter script: https://github.com/GaremoP/InverterScript
- Frontend with Angular: https://github.com/GaremoP/PV-monitoring-web
- API to fetch the data: https://github.com/GaremoP/APIsolarMonitoring

Below, are the contributions made to the repositories made throughout the development of the project. **Figure A.1** only shows those contributions made to public repositories, therefore missing some commits made when the repositories were private. Also, some of the commits to the frontend are not shown because of a mistake the first time Git was installed on the computer.

Inside each repository is a more accurate depiction of the number of commits made to each repository, the functionalities added and the dates they were done.

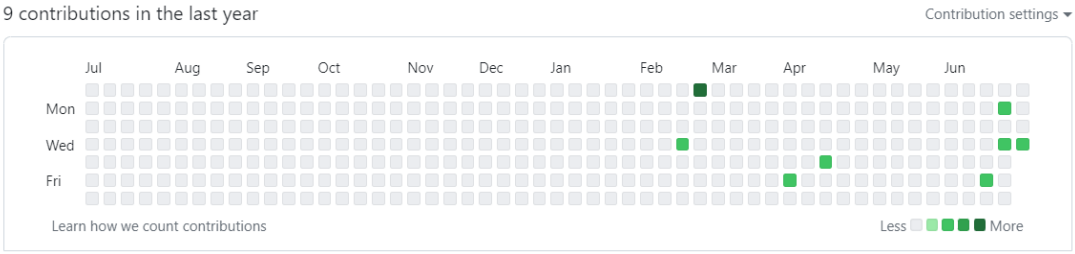


Figure A.1.- Contributions shown on Github