

SMARTLAB DIY Sensors



Author

Daniel Nuñez H. (UL)

Contact

hello@smartlablimerick.ie

Disclaimer

This document contains information that reflects only the authors' views, and the Sustainable Energy Authority of Ireland is not responsible for any use that may be made of the information it contains.

Table of Contents

Introduction.....	2
DIY Sensors.....	3
1.1 Methodology	3
1.2 DIY sensor process.....	3
1.3 DIY sensor v1	4
1.3.1 DIY sensor compact version.....	6
1.3.2 Dashboard DIY Sensor v1.....	7
1.4 DIY sensor v2.....	11
1.4.1 Dashboard DIY Sensor v2	14
1.5 DIY sensor v1 process	16
1.5.1 Electronic diagram	16
1.5.2 Sensor Code	17
1.5.3 Firebase, Database in real-time.....	21
1.5.4 Firebase Web APP.....	27
1.5.5 Web App Dashboard.....	33
1.5.6 CAD model.....	53
1.6 DIY Sensor v2 process.....	56
4.6.1 Electronic diagram	57
4.6.2 Sensor Code.....	58
4.6.3 Cayenne IoT platform.....	68
References.....	72

Introduction

The SMARTLAB project is assessing methods for implementing the Smart Readiness Indicator (SRI) within existing building stock in Limerick city centre. The SRI standard is intended to support efforts to improve efficiency, comfort and performance of buildings and contribute to targeted reductions in carbon emissions across society. The vision for this project is mapping a process where ordinarily 'non-smart' buildings can be brought up to a basic level of 'smart readiness' (O'Flaherty, 2023). The process of change encompassed by this aim is a socio-technical one, involving both technological and social innovations. A key aspect of this work is engaging with project participants to better understand how they view sensor technology and to demystify that technology so that it can be made more useful and impactful. In light of this, the project's [WP1-D1 Plan for Stakeholder Engagement](#) describes how a Do It Yourself (DIY) approach is integrated into project delivery. This work is directly related to project objectives around the empowerment of smart energy citizens and the deployment of smart infrastructure.

Although DIY sensors will not be installed as the default option in SMARTLAB participant buildings, due to ease of calibration of the factory-made devices and the need for a CE mark on the devices used, a core communication function of the project is to build and share DIY toolkits which are easy to use and understand by non-technical stakeholders (WP2, WP4). This is to support capacity building amongst stakeholders on the potential of smart building technologies and to make these technologies more accessible.

The project purchased a number of Arduino kits which can collect data critical to SRI evaluation and ongoing smart building services. There is potential for DIY sensors to contribute to an improved SRI rating in any building assessment, as it is the functionality of the technology, which is measured, and DIY sensors can potentially be more easily adapted to include control add-ons than proprietary technology. The availability of DIY kits will address several technical and financial barriers to the uptake of smart technologies already identified within project research, in the SMARTLAB deliverable [WP1-D3 Barriers to Improving the Smartness of Buildings](#).

This report outlines the work involved in developing project activities to support a Do-It-Yourself approach. It covers the development of a number of test DIY sensors for the project. It describes the 3D models made, CAD (Computer Aided Design) and ECAD (Electronic Computer Aided Design) and codes, JavaScript, HTML, C ++.

DIY Sensors

1.1 Methodology

- 1 Developing DIY (Do it yourself) devices for the SMARTLAB project must be cost-effective, easy to handle and configured for all users. The main objective of these devices is to introduce the project's end users to the Internet of Things (IoT) technology. Open-source electronic components, C++-based programming, and easily integrated sensors are used to build this device.

The device must have the following characteristics:

- Easy to install.
- Allow Wireless connection.
- Multi-parameter measurement.
 - i. Atmospheric pressure
 - ii. Light
 - iii. Temperature and relative humidity
 - iv. Particulate matter; $2.5 \mu\text{g}/\text{m}^3$
 - v. Carbon dioxide
 - vi. Volatile Organic Concentration
 - vii. Electric current
- Database.
- Data visualisation platform

The different versions of this device will follow this procedure:

1. Prototyping on a test circuit
2. Code development and work routine
3. CAD and ECAD modelling
4. Component manufacturing
5. Device construction

1.2 DIY sensor process

The DIY or Do-It-Yourself concept has become popular in recent years due to the introduction of cost-efficient technological tools, the increasing development of digital prototyping and rapid prototyping through the use of tools such as 3D printers and programmable devices (Wolf & McQuitty, 2011). Do-It-Yourself (DIY), is defined as activities in which individuals use raw and semi-raw materials and parts to produce, transform or reconstruct material possessions, including those drawn from the natural environment. (Wolf & McQuitty, 2011). This concept, which is not new, has been driven by a growing "Maker Community". The "Maker Community" is a term used to describe a community built around emerging technologies and practices that propose modifications

and rethink the use and purpose of existing materials to produce something new.(Lindtner et al., 2016).

The concept is not only around technological use but also refers to all kinds of action in which "citizen designers" create a spontaneous action that conceptualises, builds, and installs an idea. This can be applied to DIY urban interventions (Finn, 2014) or DIY artificial pancreas systems (Crabtree et al., 2019).

A DIY sensor should then be understood as a device users can build, use, and modify at will. The device to be created must comply with these elements and be cost-effective, allowing access and democratisation of technology for different users.

The concept of operation of the sensor is described in fig 1. A microcontroller will receive and send the signal from connected peripheral sensors using a Wireless network (LoRaWan, Wi-Fi, Bluetooth, Radio Frequency, 5G, GMS). The data sent must be stored in a database and with the ability to be visible through a dashboard, where the end user can observe the behaviour of their device.

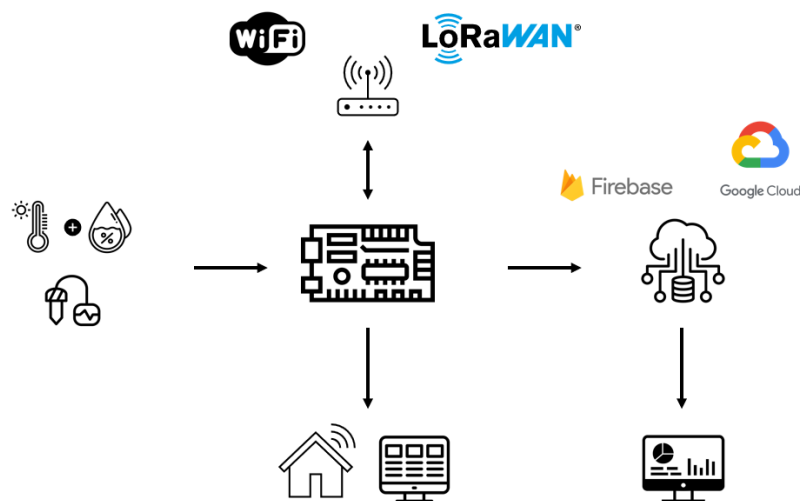
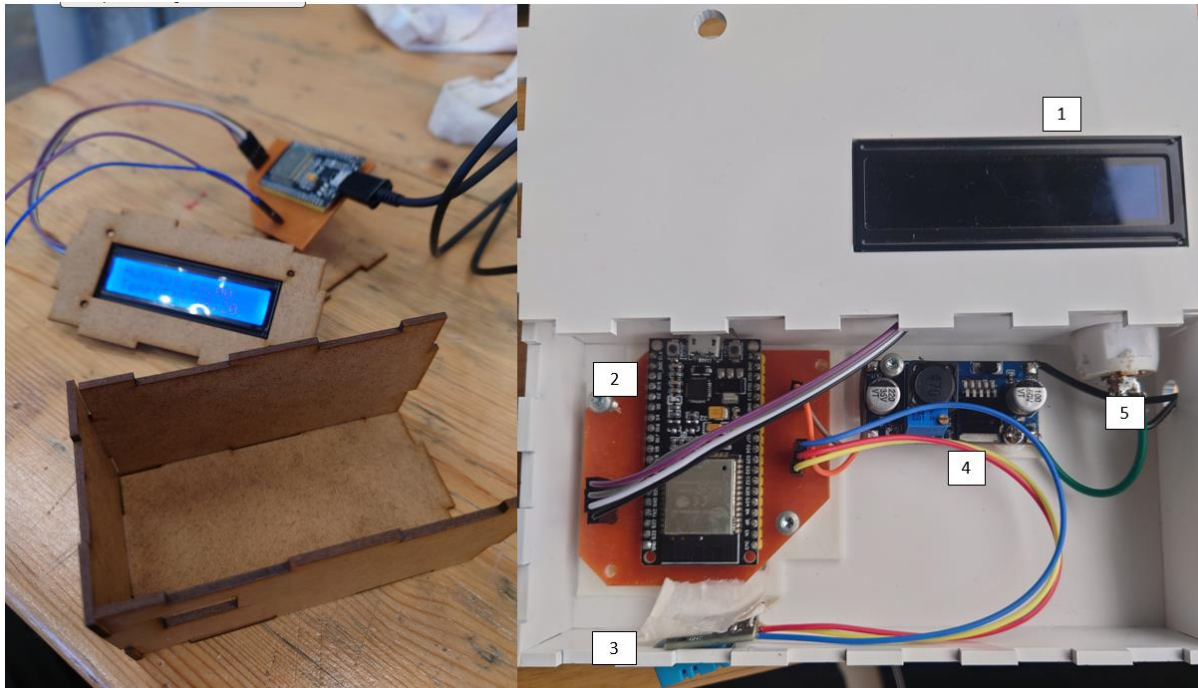


Fig. 1: DIY sensor operating scheme.

1.3 DIY sensor v1

The first version of the DIY sensors was developed between the months of October-November 2022, using elements available at FabLab Limerick at the University of Limerick. This version contains the following items.



*Fig. 2: DIY sensor v1. Prototype left image in wooden.
Right image, prototype v1.*

1. LCD Screen 16x2 / I2C
2. Microcontroller ESP 32 devkit 4
3. DHT 11 sensor
4. DC-DC converter LM2596
5. Switch on/off.

This first version of the system sends all the data acquired by the DHT 11 sensor to an IoT cloud database called Firebase.

Firebase (Google) is an app development platform whose primary function is to develop and facilitate the creation of high-quality apps quickly. It has a real-time database and the possibility of configuration for creating web apps, analyzing and using artificial intelligence [Firebase](#).



Fig. 3: DIY sensor v1. Installed in FabLab Limerick

The DIY v1 version only measures temperature and humidity in 48 data packets per day.

1.3.1 DIY sensor compact version

A simplified sensor version was developed, without an LCD screen but with the same Wireless functionality and data visualisation.

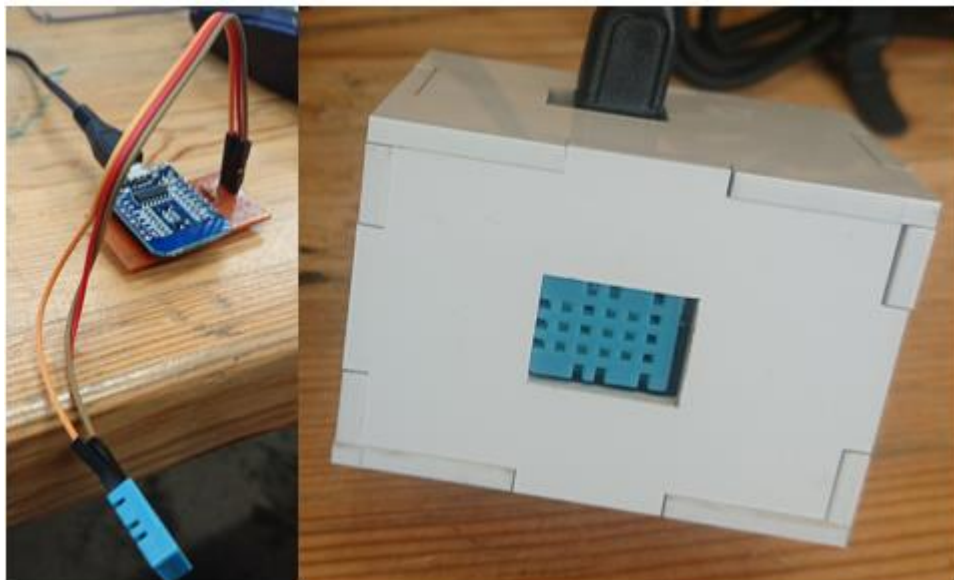


Fig. 4 DIY sensor v1 compacta

The main difference between this and version 1 is the change of the ESP32 microcontroller for an ESP2866 WEMOS. The DC converter component was also removed from the

system and replaced by a direct micro-USB connection using a universal USB charger source.

1.3.2 Dashboard DIY Sensor v1

For the first version of the sensor data visualisation, the internal ability of the ESP32 and ESP2866 microcontrollers to create a server using WEB Services was used. This was quickly modified because it did not allow for data history creation. With this in mind, a web app was developed using Firebase, a platform for IoT.

The features of this web app are the following:

1. Free to use, you only pay if you exceed the monthly data and pay by data amount.
2. It is created using a Google email account.
3. It contains real-time analysis tools and integration with artificial intelligence and Google analytics.
4. Rapid creation of functional web apps.
5. Downloads of stored data.
6. Credential configuration and data protection.

The project was hosted in a temporary link with an interface as it is shown below.

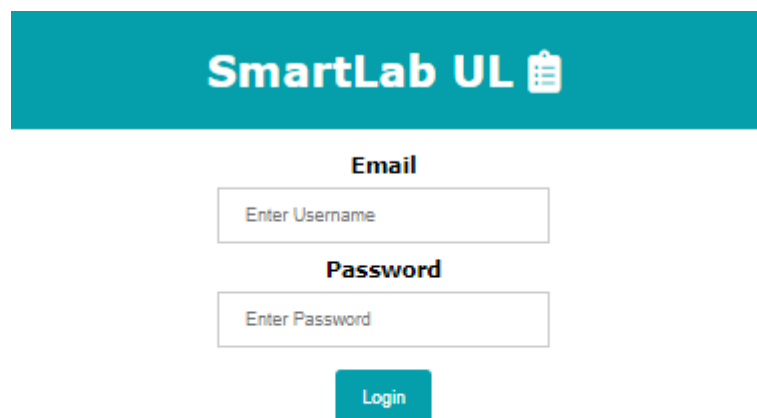
The image shows a login interface for 'SmartLab UL'. At the top is a teal header with the text 'SmartLab UL' and a clipboard icon. Below the header, the word 'Email' is centered. Underneath is a white input box with the placeholder text 'Enter Username'. Below that, the word 'Password' is centered. Underneath is another white input box with the placeholder text 'Enter Password'. At the bottom of the form is a teal button with the text 'Login' in white.

Fig. 5: Dashboard, login user.

Each development device had a data account registered in Firebase and controlled by a super-user. The credentials used for registration must be used to enter the data dashboard and view the information recorded.

Each of these users had an assigned ID. This ID is the name under which the data will be stored in the following format:

- User's ID + Temperature + Humidity + Timestamp.

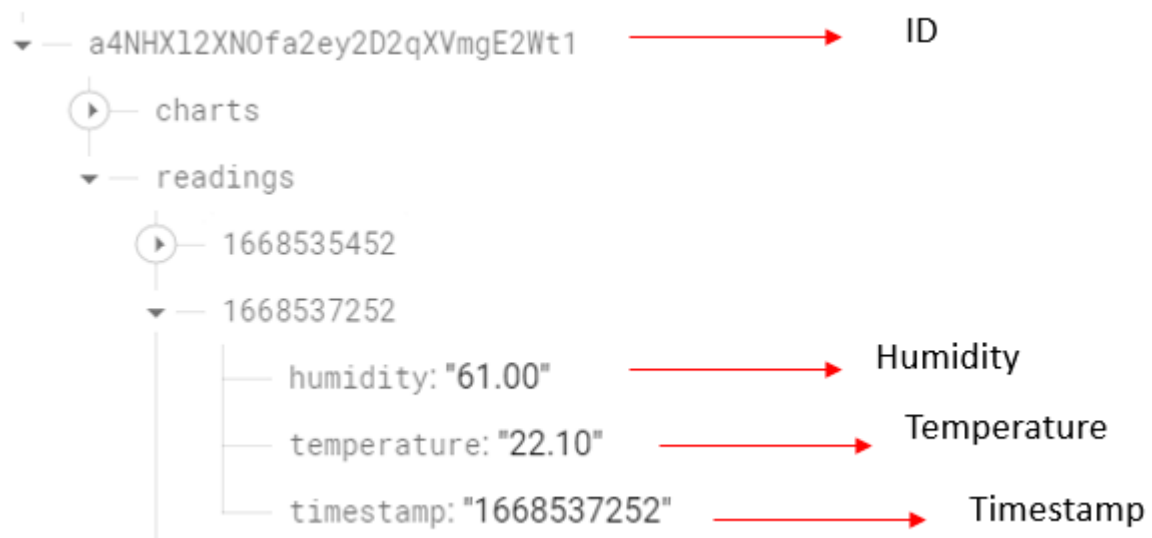


Fig. 6: Data structure.

The data generated by the sensors are entered into the database with the above structure and can be used for visualisation.

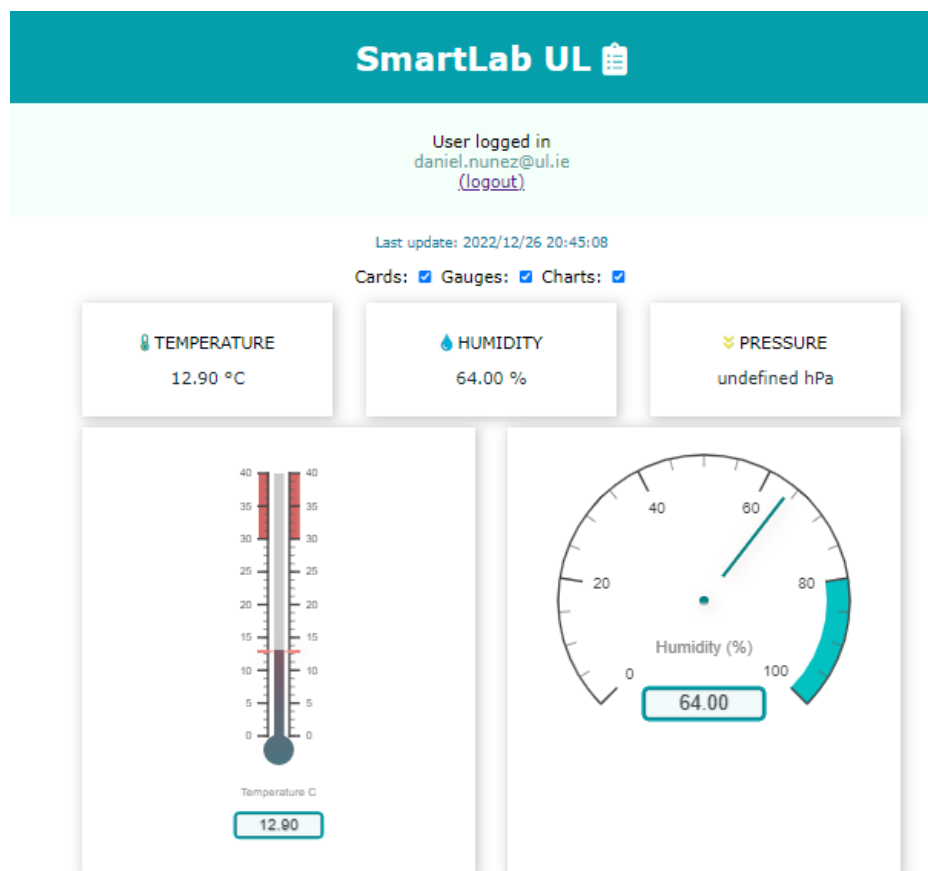


Fig. 7: Data collected by sensors in the Firebase dashboard.

The web app can be configured differently and programmed in JavaScript and HTML. With this, we can create gadgets like those seen in fig.7, graphs or tables as in fig 8.

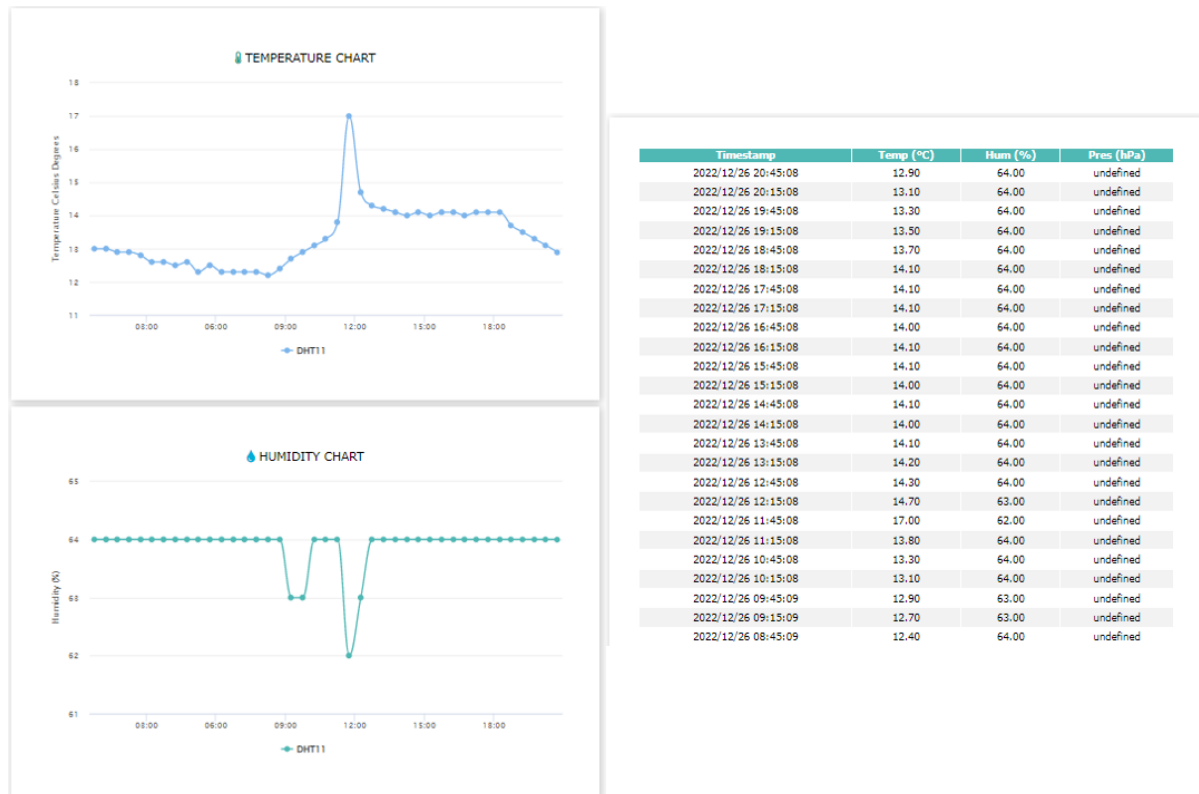


Fig. 8: Graphs and data tables in Firebase dashboard.

There are improvements to be made to this version of the dashboard, one of which is to show the historical data or desired periods. The solution was developed using PYTHON, a code that allows collecting the data generated through a document. JSON and graph the data as expected.

```
1 import json
2 import datetime
3 import matplotlib.pyplot as plt
4 import numpy as np
5 from matplotlib.colors import LinearSegmentedColormap
6
7 fig = plt.figure(figsize=(10, 10))
8 # Opening JSON file
9 f = open('smartlab_Nov_Dic.json')
10
11 # returns JSON object as
12 # a dictionary
13 data = json.load(f)
14 humedad = []
15 temperatura = []
16 time = []
17
18
19 # Iterating through the json
20 # list
21 for i in data['readings']:
22
23     t = datetime.datetime.fromtimestamp(int(data['readings'][i]['timestamp']))
24     humedad.append(data['readings'][i]['humidity'])
25     temperatura.append(float(data['readings'][i]['temperature']))
26     time.append(t)
27
28 fig = plt.axes()
29 fig.plot(time, temperatura, '-k')
30
31 plt.xticks(rotation=45, ha='right')
32
33 fig.set(xlim=(datetime.datetime(2022, 11, 15, 18, 4, 12),
34             datetime.datetime(2022, 12, 26)),
35         ylim=(7.5, 27), xlabel='Days', ylabel='Temperature',
36         title='Temperature Nov-Dic')
37
38
39 # Closing file
40 f.close()
```

Fig. 9: JSON conversion code to graph periods.

With the code in fig. 9, we can take the historical record of data saved in Firebase and observe the behaviour of the sensors. This is an example of the data obtained by the sensor hosted in FabLab from November 15, 2022, to December 26, 2022.

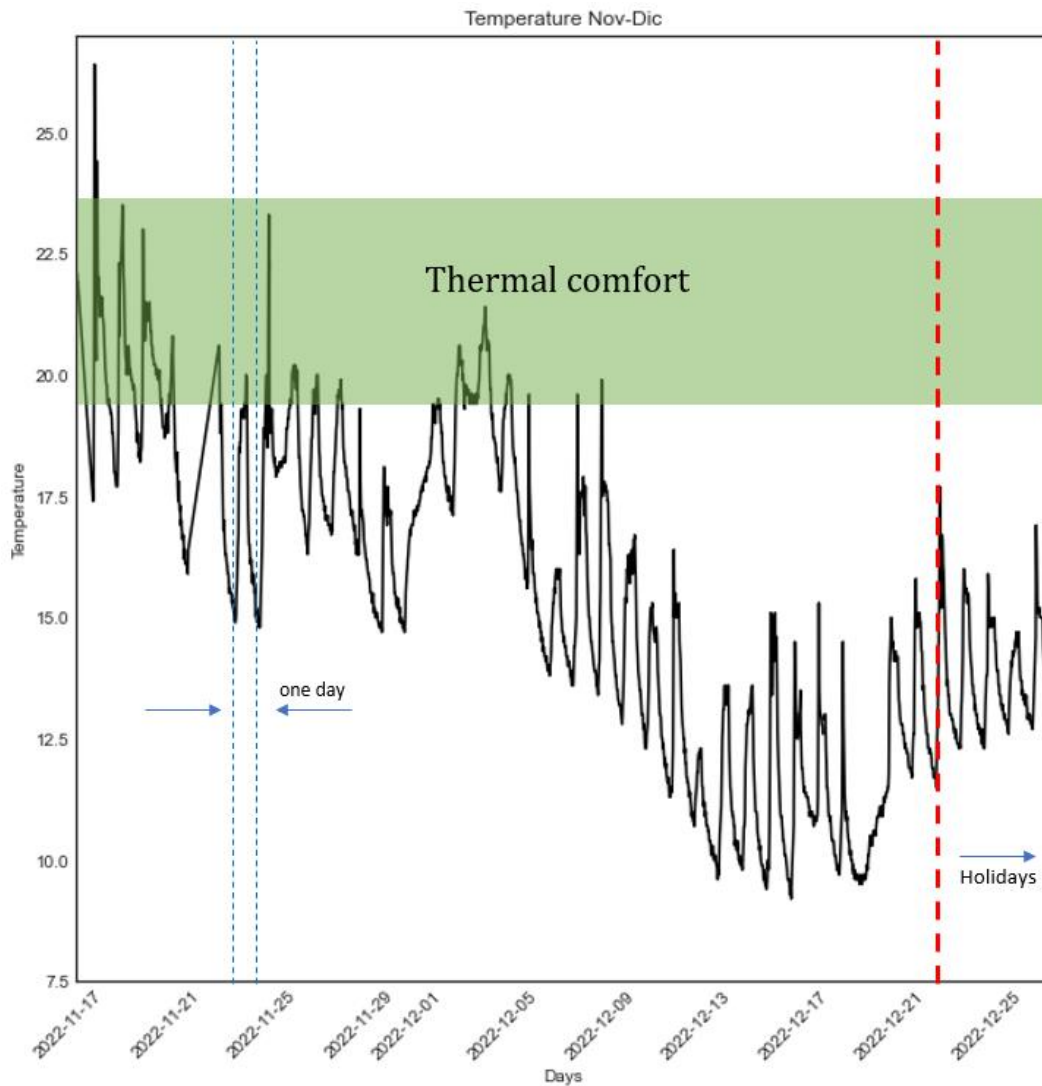


Fig. 10: FabLab thermal behaviour graph. Nov-Dec period.

1.4 DIY sensor v2

The second version of the DIY sensors is inspired by the Smart sensors of the company Milesight AM300 series with LoRawan connectivity.



Fig. 11: Smart Sensor Serie AM300.

The device has integrated 9 types of environmental measurement sensors, which are:

1. Temperature
2. Humidity
3. Movement
4. Light
5. Barometric pressure
6. TVOC
7. CO₂
8. O₂
9. Particulate matter 2.5 and 10.

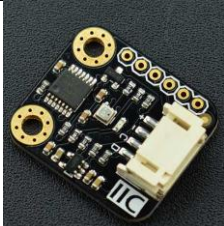


The ability to sense nine different environmental variables and transmit data through the LoRa network makes this device a powerful tool for environmental measurement in enclosed spaces.

Milesight provides technical documents and a user installation manual for the AM300 Series. The documents have the integrated sensor's characteristics and with which a list of sensors on the market with similar features is generated.

The general characteristics that must be met by the sensors to be acquired for DIY devices are the following:

- Open Source
- Plug and play
- similar precision and accuracy to the Milesight AM300 series.
- Ability to measure the same variables.

The following is the list of sensors available in the market and easy to acquire quickly.

Sensor	Variables	Image	Datasheet
BME280 Manufacturer: DFRobot	Barometric Pressure Humidity Temperature		Link
Laser PM2.5 Air Quality Sensor Manufacturer: DFRobot	PM 1.0 PM 2.5 PM 10		Link
Grove SGP 30 Seeedstudio	VOC eCO ₂		Link


Analog AC Current Sensor (20A) Manufacturer: DFRobot	Current (Max 20 A)		Link
--	--------------------	--	----------------------

Table 1. Components for DIY sensor v2.

From the list, only the current sensor is not integrated into version 2. This is because it requires an adaptation to the circuit that was not designed.

The DIY sensor v2 uses a Takachi Electric Industrial PF Series Grey ABS Enclosure, IP40, Grey Lid, 150 x 100 x 35mm. The final shape of the DIY sensor is as follows:



Fig. 12: DIY sensor v2.

1.4.1 Dashboard DIY Sensor v2

For the first version of the dashboard, Firebase was used as an IoT cloud for data collection and creating a web app for visualisation. Making such platforms requires a large number of HH to perform it correctly.

There are already designed platforms that allow the free management of visualisation blocks using MQTT or MQ Telemetry Transport communication. MQTT is a type of machine-to-machine communication, easy to implement and used in many IoT applications.

The IoT Cloud Cayenne platform from MyDevices inc. was used for this version. The platform allows the easy entry of IoT devices with LoRa, Wi-Fi, MQTT, and Ethernet connection.

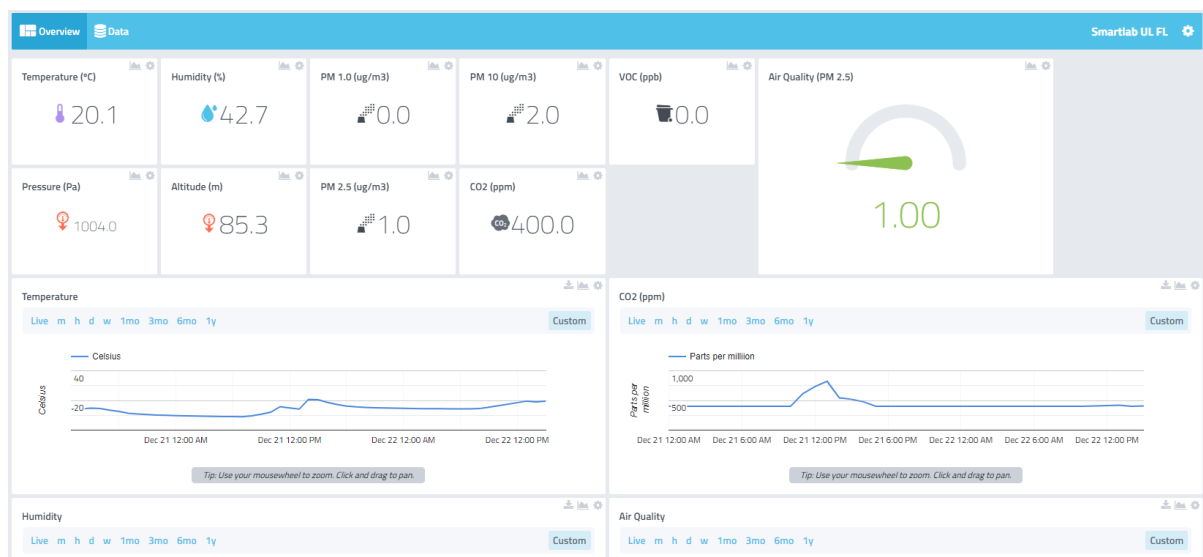


Fig. 13: DIY sensor v2.

Additionally, this platform allows the quick and easy extraction of all the data recorded by the devices.

Overview Data Smartlab UL FL							
Live m h d w 1mo Custom Query Download							
Timestamp	Device Name	Channel	Sensor Name	Sensor ID	Data Type	Unit	Values
2022-12-22 3:40:26	Smartlab UL FL	0	Temperature	08956600-806e-11ed-8d53-d7cd10251...	temp	c	20.049999237061
2022-12-22 3:40:12	Smartlab UL FL	7	VOC (ppb)	38228afo-806a-11ed-b193-d9789b2af6...	Volatile Organic	Parts per Billion	
2022-12-22 3:40:12	Smartlab UL FL	5	Air Quality	21300000-807b-11ed-b193-d9789b2af6...	Particulate Material	ug/m3	1
2022-12-22 3:40:12	Smartlab UL FL	8	CO2 (ppm)	27854dd0-8075-11ed-8d53-d7cd10251...	Carbon Dioxide	Parts per Million	400
2022-12-22 3:40:12	Smartlab UL FL	2	Altitude (m)	33ca7b70-806a-11ed-8d53-d7cd10251...			85.274002075195
2022-12-22 3:40:12	Smartlab UL FL	3	Humidity	0ce65320-807a-11ed-8d53-d7cd10251...	Humidity	%	42.651000976562
2022-12-22 3:40:12	Smartlab UL FL	4	PM 1.0 (ug/m3)	35bc8770-806a-11ed-8d53-d7cd10251...			
2022-12-22 3:40:12	Smartlab UL FL	6	PM 10 (ug/m3)	3799d2f0-806a-11ed-8d53-d7cd10251...			2
2022-12-22 3:40:12	Smartlab UL FL	1	Pressure (Pa)	331694c0-806a-11ed-8d53-d7cd10251...	bp	hpa	1003.9699707031
2022-12-22 3:40:11	Smartlab UL FL	0	Temperature	08956600-806e-11ed-8d53-d7cd10251...	temp	c	20.079999923706
2022-12-22 3:39:57	Smartlab UL FL	6	PM 10 (ug/m3)	3799d2f0-806a-11ed-8d53-d7cd10251...			2
2022-12-22 3:39:57	Smartlab UL FL	3	Humidity	0ce65320-807a-11ed-8d53-d7cd10251...	Humidity	%	42.618000030518
2022-12-22 3:39:57	Smartlab UL FL	2	Altitude (m)	33ca7b70-806a-11ed-8d53-d7cd10251...			85.274002075195
2022-12-22 3:39:57	Smartlab UL FL	4	PM 1.0 (ug/m3)	35bc8770-806a-11ed-8d53-d7cd10251...			
2022-12-22 3:39:57	Smartlab UL FL	8	CO2 (ppm)	27854dd0-8075-11ed-8d53-d7cd10251...	Carbon Dioxide	Parts per Million	400
2022-12-22 3:39:57	Smartlab UL FL	7	VOC (ppb)	38228afo-806a-11ed-b193-d9789b2af6...	Volatile Organic	Parts per Billion	
2022-12-22 3:39:57	Smartlab UL FL	5	Air Quality	21300000-807b-11ed-b193-d9789b2af6...	Particulate Material	ug/m3	1
2022-12-22 3:39:57	Smartlab UL FL	1	Pressure (Pa)	331694c0-806a-11ed-8d53-d7cd10251...	bp	hpa	1003.950012207
2022-12-22 3:39:56	Smartlab UL FL	0	Temperature	08956600-806e-11ed-8d53-d7cd10251...	temp	c	20.10000038147
2022-12-22 3:39:41	Smartlab UL FL	7	VOC (ppb)	38228afo-806a-11ed-b193-d9789b2af6...	Volatile Organic	Parts per Billion	
2022-12-22 3:39:41	Smartlab UL FL	8	CO2 (ppm)	27854dd0-8075-11ed-8d53-d7cd10251...	Carbon Dioxide	Parts per Million	400
2022-12-22 3:39:41	Smartlab UL FL	4	PM 1.0 (ug/m3)	35bc8770-806a-11ed-8d53-d7cd10251...			
2022-12-22 3:39:41	Smartlab UL FL	3	Humidity	0ce65320-807a-11ed-8d53-d7cd10251...	Humidity	%	42.575000762939
2022-12-22 3:39:41	Smartlab UL FL	6	PM 10 (ug/m3)	3799d2f0-806a-11ed-8d53-d7cd10251...			2

Fig. 14: DIY sensor v2.

The platform allows the entry of different users, and these, in turn, visualise only the information of the devices appended with their ID. They will enable each user to manipulate and analyse the data that personal devices generate.

1.5 DIY sensor v1 process

This section is a detailed guide to the process of creating DIY sensors. The codes, CAD, eCAD and 3D models made will be described. The points to detail are the following:

- Code for compact v1, temperature and humidity sensor, DHT 11.
- CAD, eCAD and 3D models are used for circuit design, internal components and enclosure.
- Codigo creación firebase's database and dashboard

The DIY sensor v1 was created using the following components:

- MicroUSB Cable
- ESP8266 Wemos Microcontroller
- DHT11
- Acrilico or case enclosure ABS

These components can measure the minimum variables, temperature and humidity in an indoor space.

1.5.1 Electronic diagram

The following schematic was developed in Autodesk Eagle for connecting the components to the microcontroller.

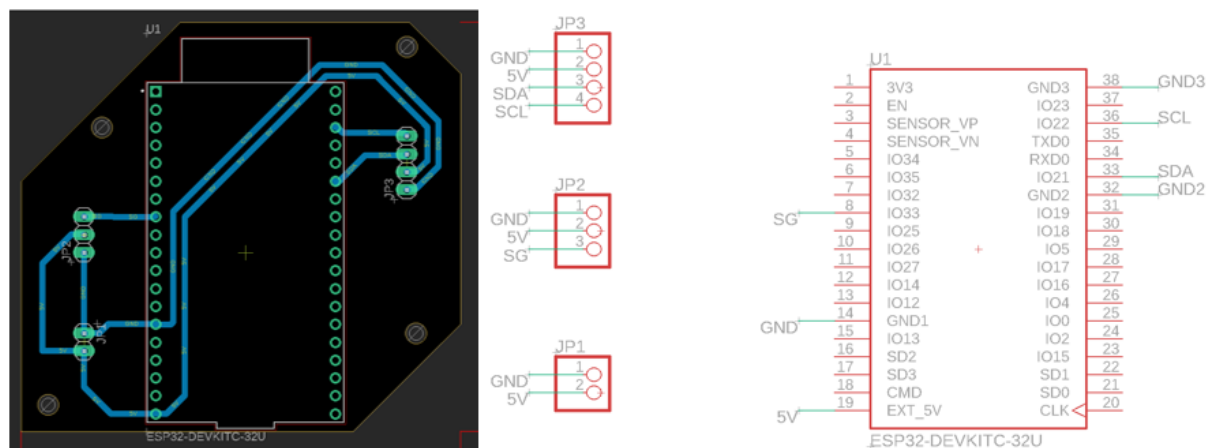


Fig. 15: Electronic Schematic DIY sensor v1

For version v1, the ESP32 dev kit 4 component is used as a microcontroller. It picks up the signal from the DHT11 sensor, allowing temperature and humidity readings. Additionally, it has a 16x2 I2C LCD screen for data visualisation.

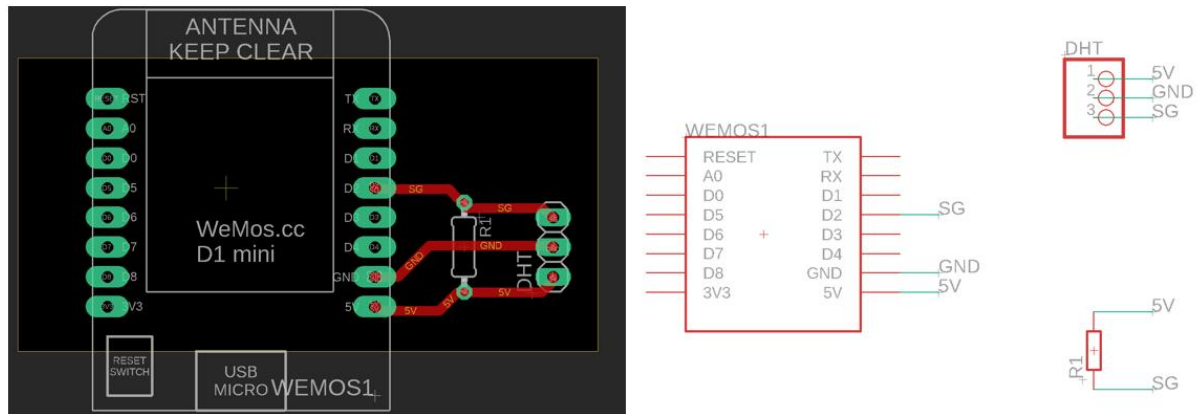


Fig. 16: Electronic Schematic DIY sensor v1 compact

In fig 7.1.1. 2, we have the v1 compact version. The main component is the ESP8266 wemos, the microcontroller of the sensor. It is the one that controls the signals and allows the collection and visualise the information generated by the DTH11 sensor.

The DTH sensor is connected to the digital PIN D2 of the microcontroller. Additionally, a 10k ohm resistor must be attached to stabilise the signal coming from the sensor.

1.5.2 Sensor Code

The code of sensor V1 compact is in the file with the name ESP_Officev1_Firebase_ESP8266_. The code is as follows:

```
#include <Arduino.h>
#include <ESP8266WiFi.h>
#include <Firebase_ESP_Client.h>
#include <Wire.h>
#include <NTPClient.h>
#include <WiFiUdp.h>
#include "DHT.h"
#define DHTPIN D2 // Digital pin connected to the DHT sensor
#define DHTTYPE DHT11 // DHT 11

// Provide the token generation process info.
#include "addons/TokenHelper.h"
// Provide the RTDB payload printing info and other helper functions.
#include "addons/RTDBHelper.h"

// Insert your network credentials
#define WIFI_SSID "Wi-Fi Name"
#define WIFI_PASSWORD "Wi-Fi Password"
```

Libraries necessary for the operation of the code

```
// Insert Firebase project API Key
#define API_KEY "AlzaSyANbD-63734Mde4wGdyFrZeZMkl6HLsqxU"

// Insert Authorised Email and Corresponding Password
#define USER_EMAIL "User email"
#define USER_PASSWORD "User password"

// Insert RTDB URL define the RTDB URL
#define DATABASE_URL "https://SMARTLAB-54c95-default-rtdb.europe-west1.firebaseio.app/"

// Define Firebase objects
FirebaseData fbdo;
FirebaseAuth auth;
FirebaseConfig config;

// Variable to save USER UID
String uid;

// Database main path (to be updated in setup with the user UID)
String databasePath;
// Database child nodes
String tempPath = "/temperature";
String humPath = "/humidity";
String presPath = "/pressure";
String timePath = "/timestamp";

// Parent Node (to be updated in every loop)
String parentPath;

FirebaseJson json;

//DHT sensor config
DHT dht(DHTPIN, DHTTYPE);

String TemperatureDHT11() {
    float t = dht.readTemperature();
    if (isnan(t)) {
        Serial.println("Failed to read from DHT11 sensor!");
        return "";
    }
    else {
        Serial.println(t);
        return String(t);
    }
}

String HumidityDHT11() {
```

KEY API generated by the firebase database. Send the data to this address

It must be equal to the one registered in

Define objects to use to send data and logging

Text variables type String, will serve to organize the data

The format in which Json-type data is sent.

Function to call the DHT reading, temperature and humidity

```

float h = dht. readHumidity();
if (isnan(h)) {
    Serial. println("Failed to read from DHT11 sensor!" );
    return "";
}
else {
    Serial, serial. println(h);
    return String(h);
}
}

// Timer variables (send new readings every three minutes)
unsigned long sendDataPrevMillis = 0;
unsigned long timerDelay = 1800000;

// Initialize WiFi
void initWiFi() {
    Wi-Fi. begin(WIFI_SSID, WIFI_PASSWORD);
    Serial. print("Connecting to WiFi .." );
    while (WiFi.status() != WL_CONNECTED) {
        Serial. print('.');
        delay(1000);
    }
    Serial, serial. println(WiFi. localIP());
    Serial, serial. println();
}

// Define NTP Client to get time
WiFiUDP ntpUDP;
NTPClient timeClient(ntpUDP, "pool.ntp.org");

// Variable to save current epoch time
int timestamp;

//function that gets current epoch time
unsigned long getTime() {
    timeClient. update();
    unsigned long now = timeClient. getEpochTime();
    return now;
}

void setup(){
    Serial.begin(115200);
    dht. begin();

    initWiFi();
    timeClient. begin();

```

Function to call the DHT reading, temperature and humidity

It sends data every 1.8 M milliseconds = 30 min.

Function for Wifi connection

Function to have the time and date according to the Location

Start the operation by calling all functions to start the system.
This process is only done when turned on

```
// Assign the api key (required)
config. api_key = API_KEY;

// Assign the user sign in credentials
auth. user. email = USER_EMAIL;
auth. user. password = USER_PASSWORD;

// Assign the RTDB URL (required)
config. database_url = DATABASE_URL;

Firebase. reconnectWiFi(true);
fbdo. setResponseSize(4096);

// Assign the callback function for the long running token generation task */
config. token_status_callback = tokenStatusCallback; //see addons/TokenHelper.h

// Assign the maximum retry of token generation
config. max_token_generation_retry = 5;

//Initialise the library with the Firebase authen and config
Firebase. begin(&config, &auth);

// Getting the user UID might take a few seconds
Serial. println("Getting User UID");
while ((auth. token. uid) == "") {
  Serial.print('.');
  delay(1000);
}
// Print user UID
uid = auth. token. uid. c_str();
Serial. print("User UID: ");
Serial. println(uid);

// Update database path
databasePath = "/UsersData/" + uid + "/readings";
}

void loop(){

// Send new readings to database
if (Firebase.ready() && (millis() - sendDataPrevMillis > timerDelay || sendDataPrevMillis == 0)){
  sendDataPrevMillis = millis();

//Get current timestamp
timestamp = getTime();
Serial. print ("time: ");
Serial, serial. println(timestamp);
```

Use the user's credentials to generate a data ID. Thus, this computer is recognizable in the database.

Finally, the generated data will be recorded in a folder called readings.

Start the Read Code, sending a data and starting to count 30 min

```
parentPath= databasePath + "/" + String(timestamp);

json. set(tempPath. c_str(), String(dht. readTemperature()));
json. set(humPath. c_str(), String(dht. readHumidity()));
// json.set(presPath.c_str(), String(bme.readPressure()/100.0F));
json. set(timePath, String(timestamp));
Serial. printf("Set json... %s\n", Firebase. RTDB. setJSON(&fbdo, parentPath. c_str(), &json) ?
"ok" : fbdo. errorReason(). c_str());
}
}
```

Every time 30 minutes pass, the system will take the time and date, more temperature and humidity of the moment and send them to firebase.

The code is quite simple and makes up Firebase's main structure of data generation. Next, proceed to the explanation of the creation of the database as well as the registration of users.

1.5.3 Firebase, Database in real-time.

To create a database in Firebase, you need only one Google account. Once the account is entered, we will have a front page similar to the following figure.

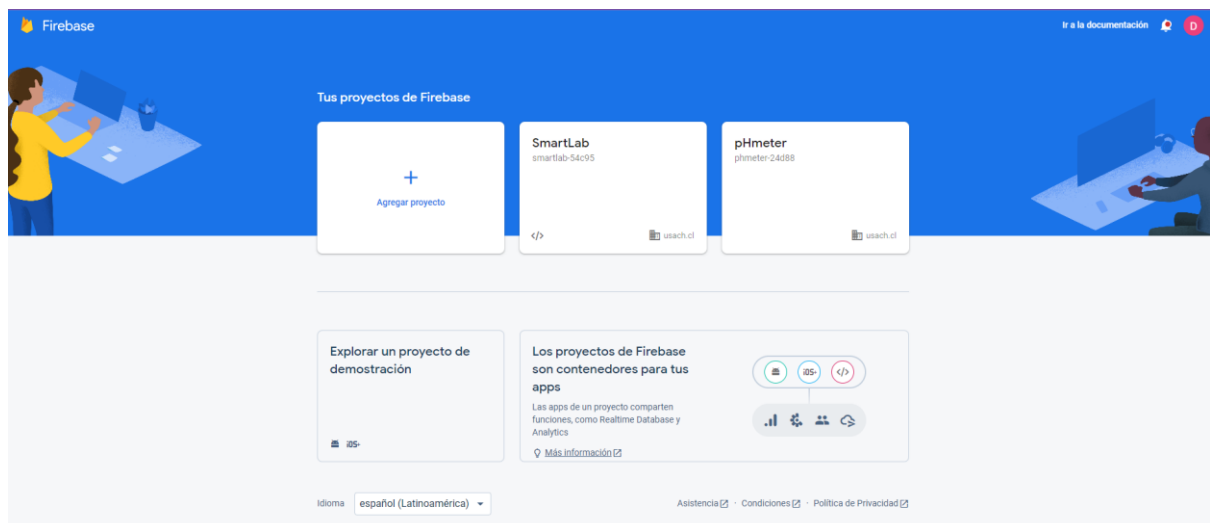


Fig. 17: Firebase Front page.

Once on this front, we start creating a new project.

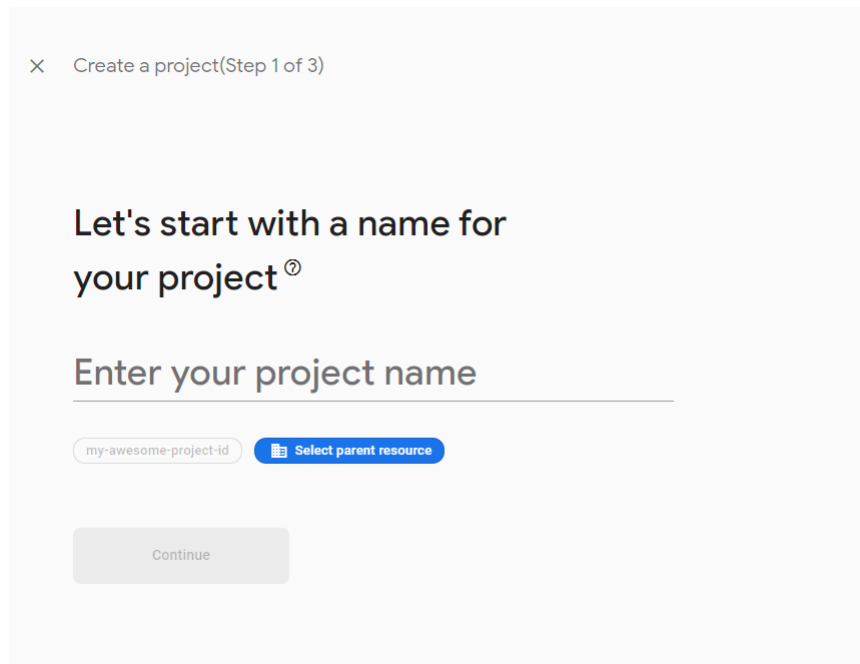


Fig. 18: Database name

Once the creation of the project is finished, in the process, it will ask about the product of Google Analytics, and we will proceed to the creation of credentials for our database.

1. We proceed to the configuration of authentication methods.

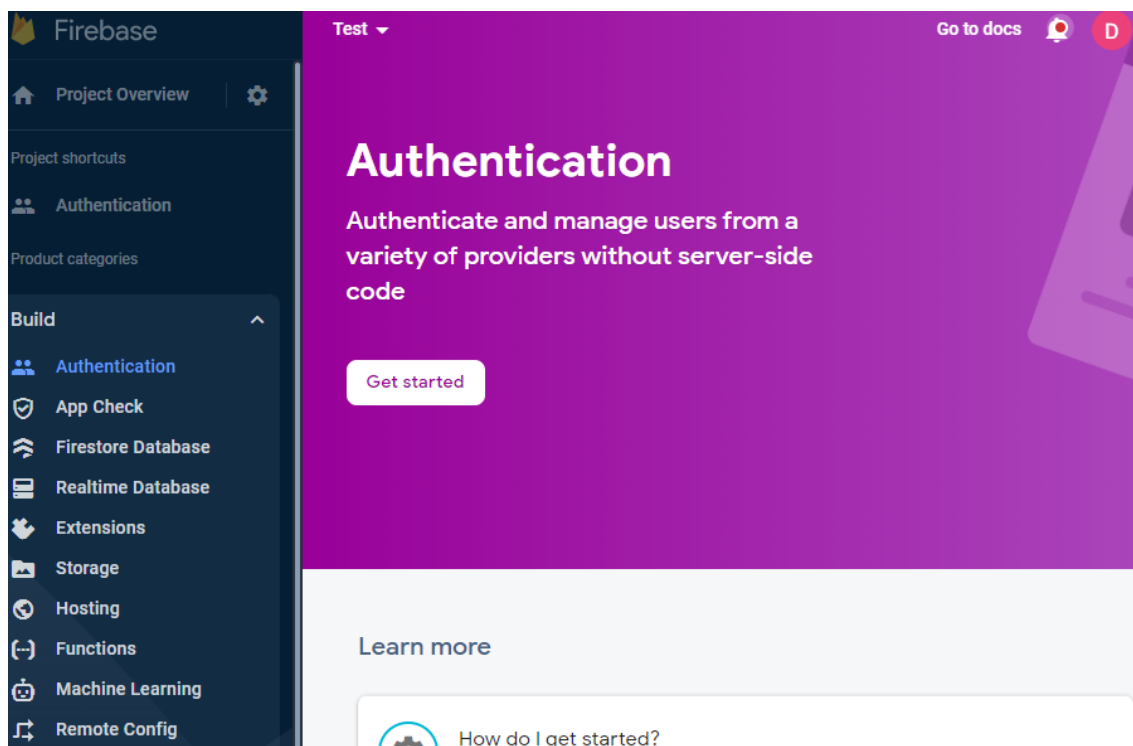


Fig. 19: Database authentication methods

There are several options to register rear. For the prototype, we have used the option, Email / Password. This allows users to create without necessarily having a Google account. It can use Google, Facebook, and Twitter accounts, among others.

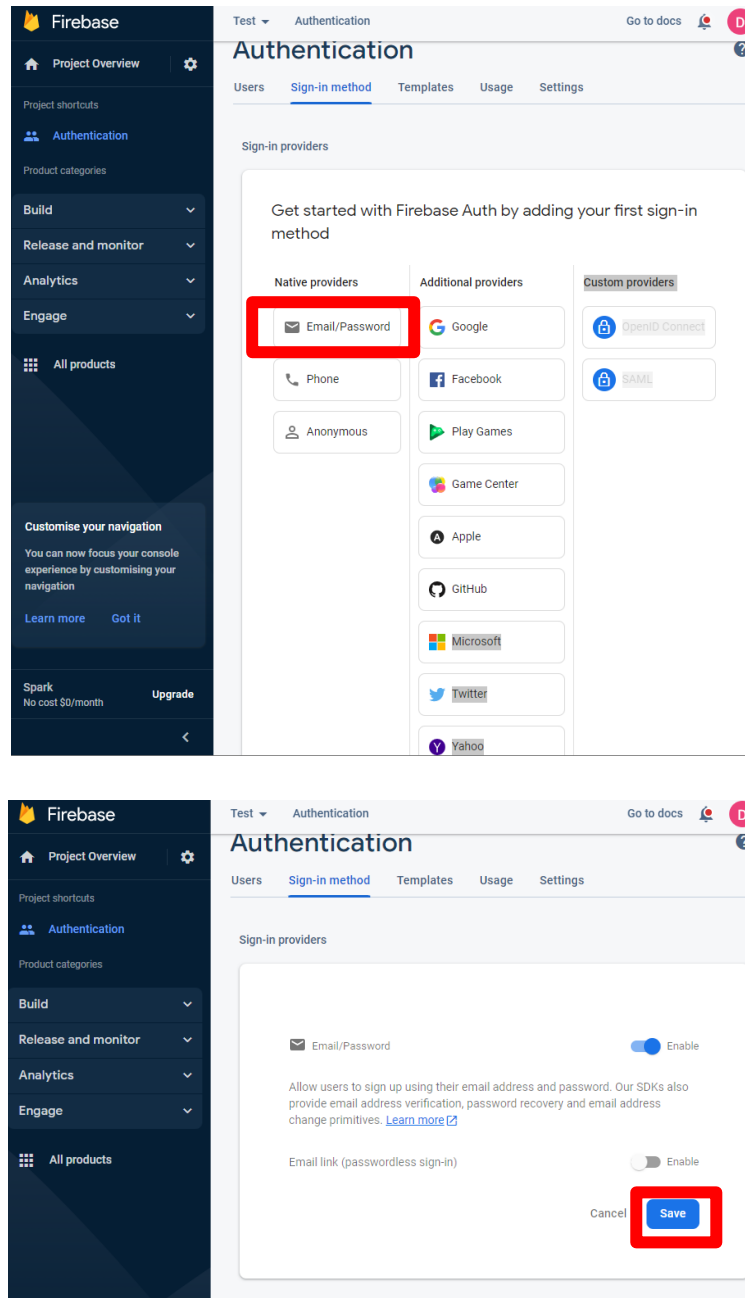


Fig. 20: Database authentication methods Email/Password.

2. Once the process is finished, email and password should be enabled. Users are then added.

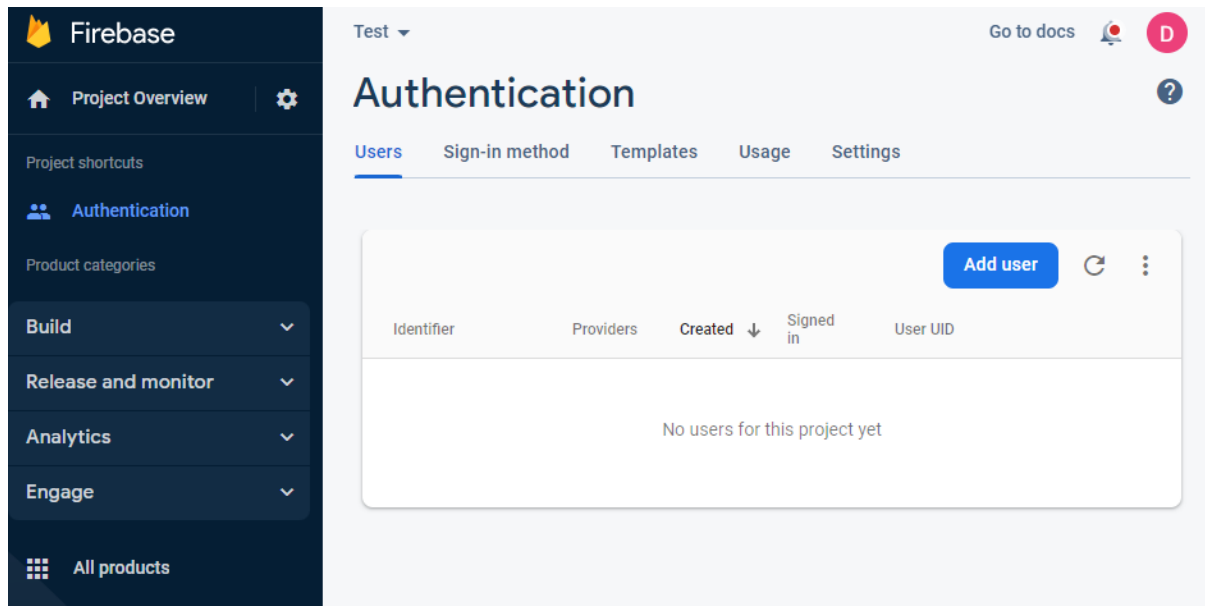


Fig. 21: Database authentication methods Add users.

- When users are created, a UID is automatically created. DIY sensors use these UIDs to store users' environmental data.

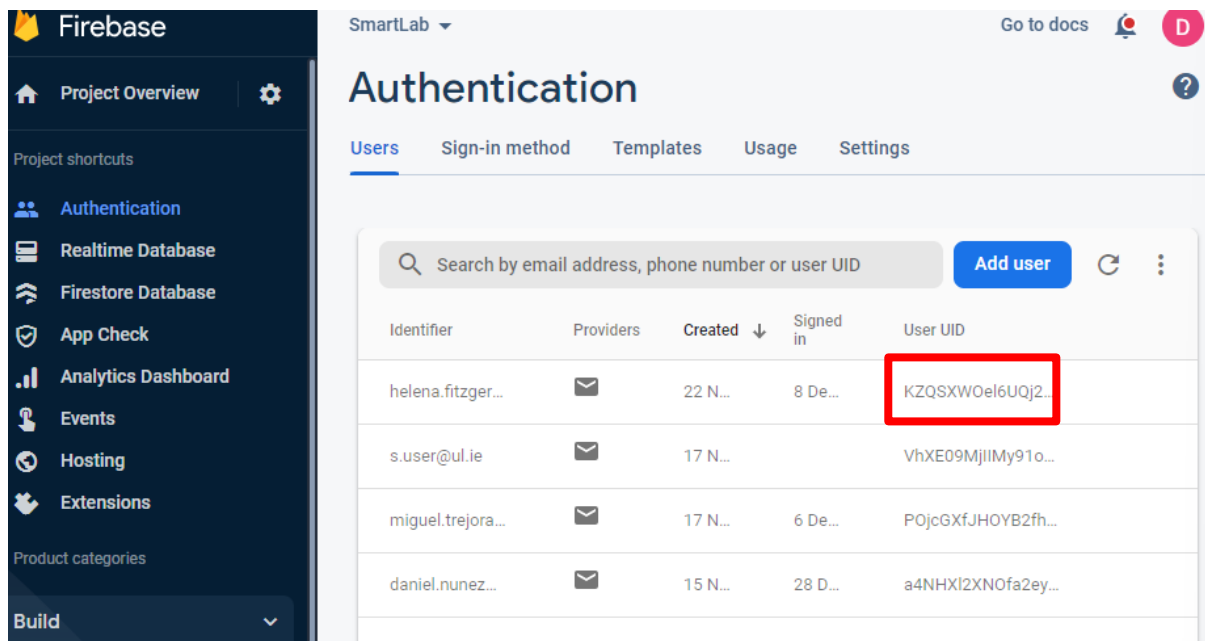


Fig. 22: Database authentication methods UID

- With this, our database already has a user registration mode; therefore, we can send personalised data. We now need a key to allow different teams to access the API KEY project. For this, we will go to the setting.

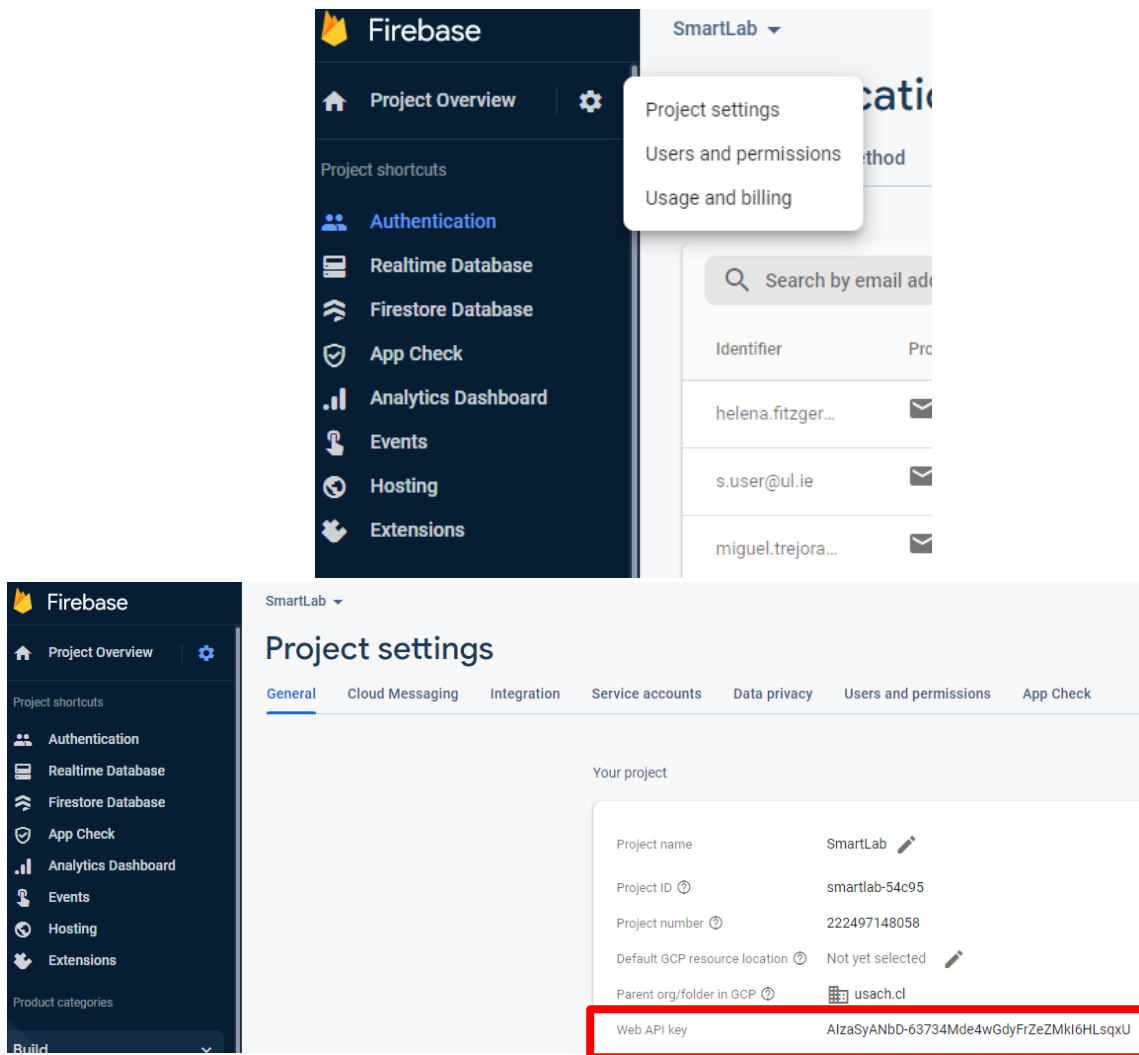
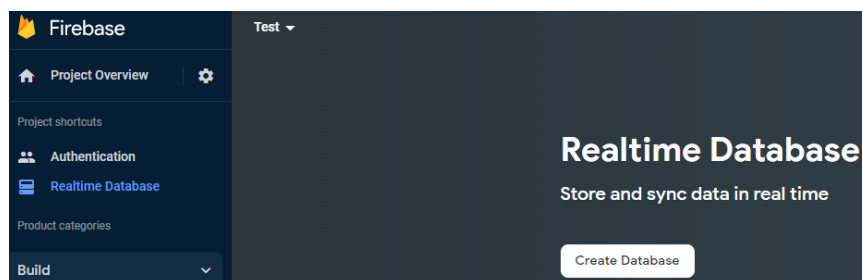


Fig. 23: Database API KEY process

The API KEY is required for the data to be sent to our database. In section 3.5.2, the code description notes where the KEY API should be considered.

5. Once the KEY API is obtained, the Realtime Database is configured. Our database will be hosted on the server closest to our location. For our prototype, Belgium (Europe-west1).



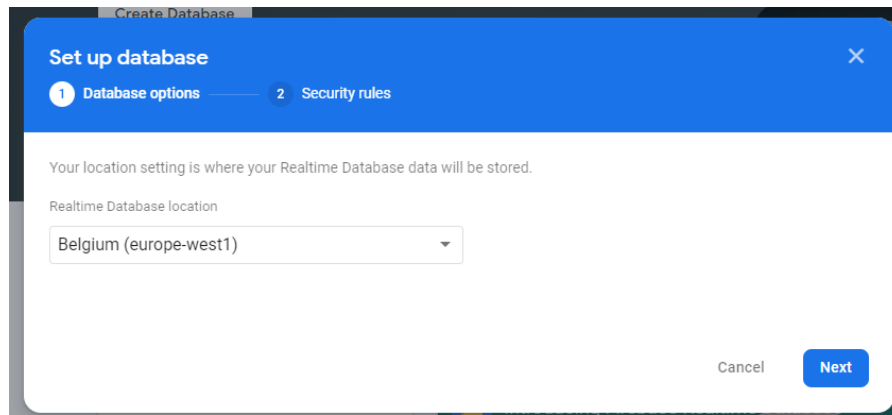


Fig. 24: Set up Real-time Database.

6. In security rules, we must allow data entry and visualisation by third parties. This must be configured later to establish rules that protect the data. As it is a prototype, we will leave the safety rules in test mode.

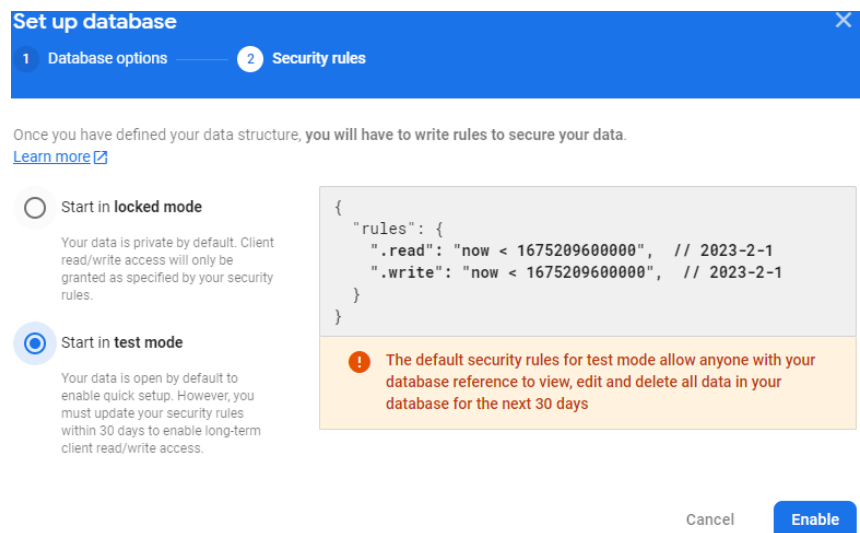


Fig. 25: Set up Real-time Database and security rules.

7. Once this process is done, our database is configured and ready to receive data. We will only add one security rule to our database. Only registered users will have permission to edit or view the data in it. You must copy the following code and place it in the security rules.

```

8. // These rules grant access to a node matching the authenticated
9. // user's ID from the Firebase auth token
10. {
11.   "rules": {
12.     "UsersData": {
13.       "$uid": {
14.         ".read": "$uid === auth.uid",

```

```
15.   ".write": "$uid === auth.uid"
16.   }
17. }
18. }
19. }
```

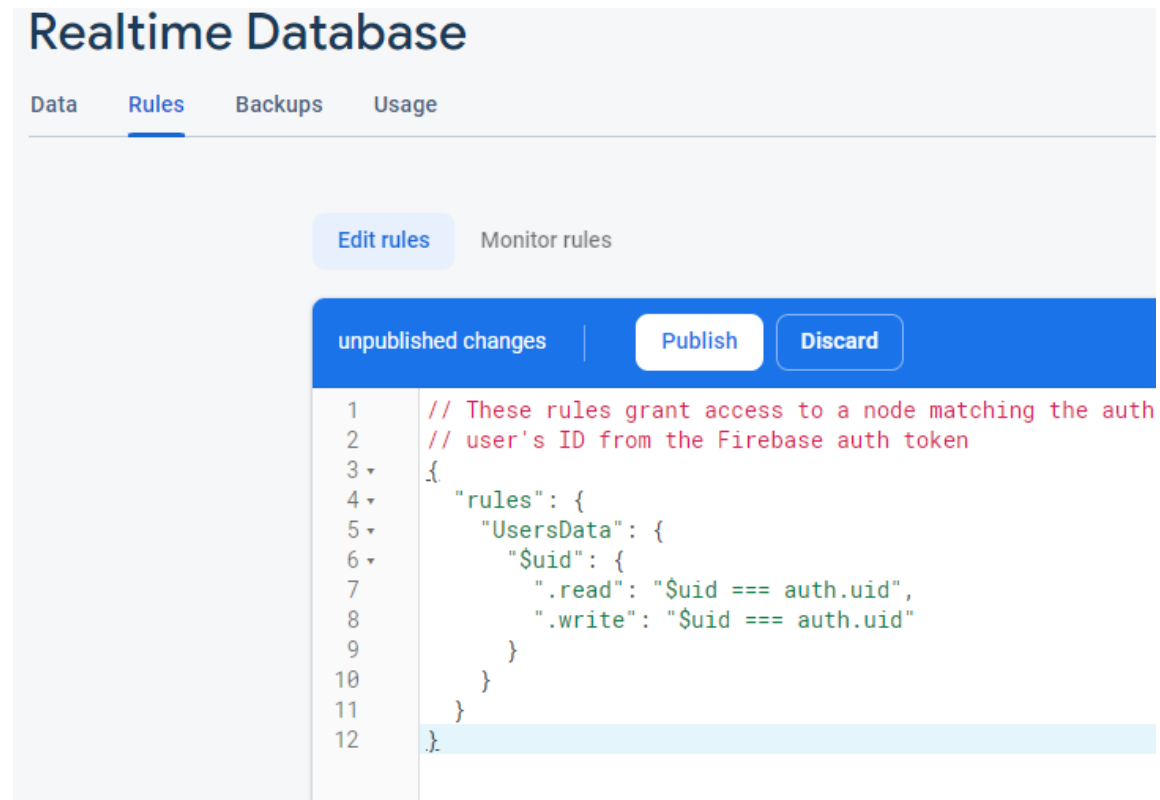


Fig. 26: User UID, security rules.

The database configuration is now complete.

1.5.4 Firebase Web APP

We will proceed to the creation of the dashboard app in Firebase, which has the following features:

- SSL certificate
- Firebase Hosting server
- Authentication process
- Web app to visualise the data obtained by the sensors. The visualisation is done in graphs, gauges or tables.
- All data is restricted from use by database security rules.

To create our dashboard, we will need to install free software to design our Web App. The software to be installed is:

- Visual Studio Code
- Node.JS LTS version
- Install Node.js Extension Pack (VS Code)
- Install Firebase tools (VS Code)

There are several tutorials on the internet on how to install this software correctly and how to create a web app in Firebase. This document will only explain the configurations made to observe and visualise the data in the web app.

Let's start with the process of creating the SMARTLAB Web App.

1. With our database created, we must now add an APP that allows us to visualise the data. For this, we must go to our project in Firebase.

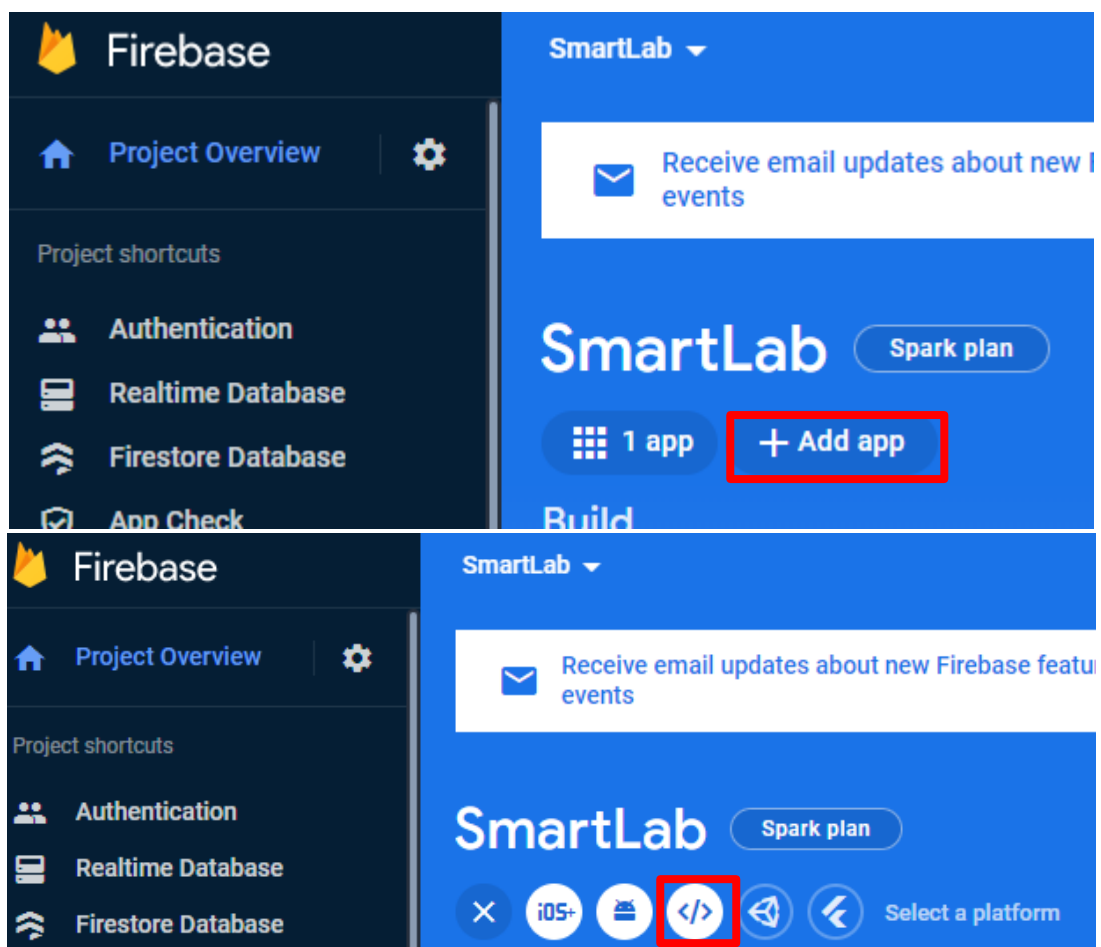
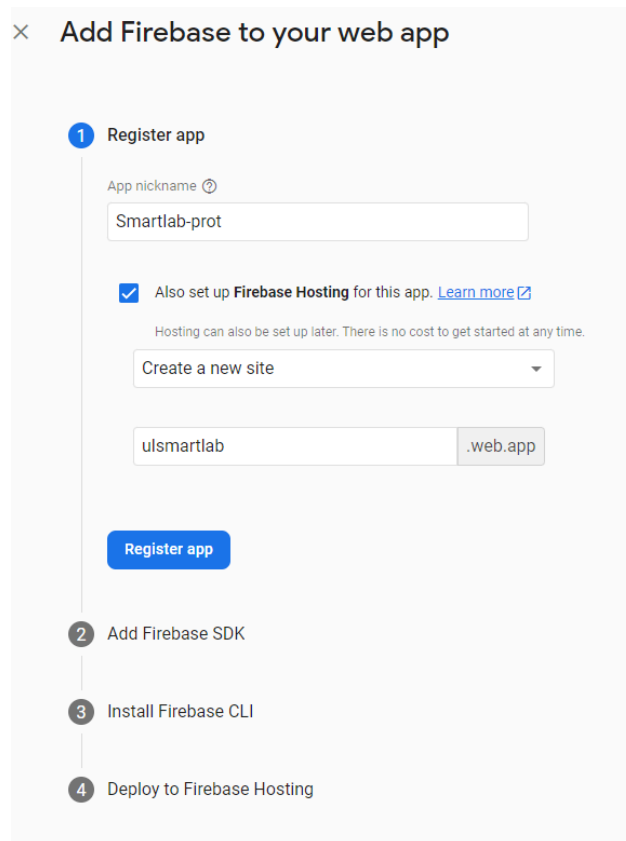


Fig. 27: Add APP Firebase Project.

2. We completed the registration form for our web app.



The screenshot shows a web form titled "Add Firebase to your web app" with a close button (X) in the top left. The form is divided into four numbered steps: 1. Register app, 2. Add Firebase SDK, 3. Install Firebase CLI, and 4. Deploy to Firebase Hosting. Step 1 is the active step. It contains a text input for "App nickname" with the value "Smartlab-prot". Below this is a checkbox labeled "Also set up Firebase Hosting for this app." which is checked, followed by a link "Learn more". A note states "Hosting can also be set up later. There is no cost to get started at any time." Below this is a dropdown menu labeled "Create a new site" with a downward arrow. At the bottom of step 1 is a blue button labeled "Register app".

Fig. 28: Form Firebase Web App.

3. The most important thing about this process is to get our SDK. This element allows us to access our database and create configurations using HTML and JAVA.

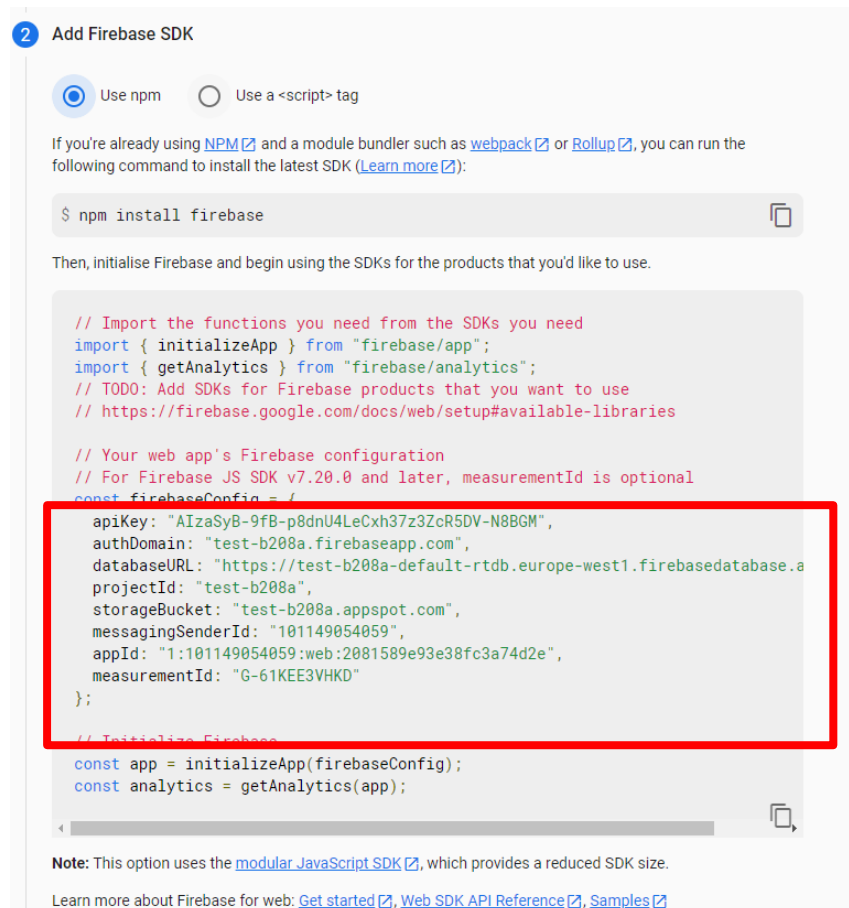


Fig. 29: Firebase Web App SDK

Once the creation form is completed and our SDK obtained, we proceed to the creation of the web app using Visual Studio Code.

1. The first thing to do is create a Web app project folder. Office device is my folder for the SMARTLAB project.
2. Open VS Code, file > open folder and select the project folder.
3. We open a terminal in VS Code and should write "firebase login."

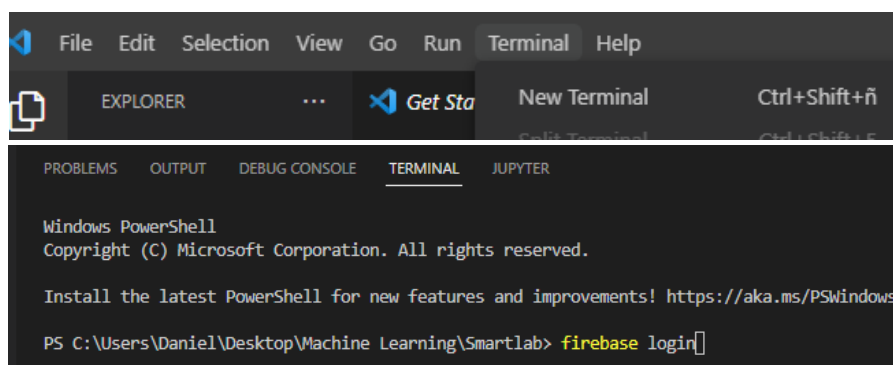


Fig. 30: Firebase VSC Installation

- This action should allow us to log in to our firebase account and use our projects created there.

```
Already logged in as daniel.nunezh@usach.cl
PS C:\Users\Daniel\Desktop\Machine Learning\Smartlab>
```

Fig. 31: Firebase VSC Installation part I

- The next step is to launch the web app. For this, we write "firebase init" from the same terminal and proceed (Y).

```
PS C:\Users\Daniel\Desktop\Machine Learning\Smartlab> firebase init

##### 
##      ##      ##      ##      ##      ##      ##      ##      ##      ##
#####  ##  #####  #####  #####  #####  #####  #####  #####  #####
##      ##  ##      ##      ##      ##      ##      ##      ##      ##
##      #####  #####  #####  #####  #####  #####  #####  #####
##      ##      ##      ##      ##      ##      ##      ##      ##      ##
##      #####  ##  #####  #####  ##      ##  #####  #####

You're about to initialize a Firebase project in this directory:

C:\Users\Daniel\Desktop\Machine Learning\Smartlab

? Are you ready to proceed? (Y/n)
```

Fig. 32: Firebase VSC Installation part II

- Select with the "Space" key the options "Real-time Database" and "Hosting: Configure files for Firebase Hosting...".

```
? Which Firebase features do you want to set up for this directory? Press Space to select features
(*) Realtime Database: Configure a security rules file for Realtime Database and (optionally) provide
( ) Firestore: Configure security rules and indexes files for Firestore
( ) Functions: Configure a Cloud Functions directory and its files
>(*) Hosting: Configure files for Firebase Hosting and (optionally) set up GitHub Action deploys
( ) Hosting: Set up GitHub Action deploys
( ) Storage: Configure a security rules file for Cloud Storage
( ) Emulators: Set up local emulators for Firebase products
(Move up and down to reveal more choices)
```

Fig. 33: Firebase VSC Installation part III

- We use an existing project, as we will use the project we already created in Firebase. Select and press "Enter".

```
? Please select an option: (Use arrow keys)
> Use an existing project
  Create a new project

? Please select an option: Use an existing project
? Select a default Firebase project for this directory:
> phmeter-24d88 (pHmeter)
  smartlab-54c95 (SmartLab)
  test-b208a (Test)
```

Fig. 34: Firebase VSC Installation part IV

- Before the question "What file should be used for Realtime Database Security Rules?" press "Enter" to automatically select the security rules already written

above. Additionally, the following questions will appear, and we will give the answers in blue:

- What do you want to use as your public directory? **Public**
- Configure as a single-page app (rewrite all URLs to /index.html). **No**
- Set up automatic builds and deploy them with GitHub? **No**

```
? What file should be used for Realtime Database Security Rules? database.rules.json
+ Database Rules for test-b208a-default-rtdb have been written to database.rules.json.
Future modifications to database.rules.json will update Realtime Database Security Rules when you run
firebase deploy.

=== Hosting Setup

Your public directory is the folder (relative to your project directory) that
will contain Hosting assets to be uploaded with firebase deploy. If you
have a build process for your assets, use your build's output directory.

? What do you want to use as your public directory? public
? Configure as a single-page app (rewrite all urls to /index.html)? No
? Set up automatic builds and deploys with GitHub? No
+ Wrote public/404.html
+ Wrote public/index.html
```

Fig. 35: Firebase VSC Installation part V

9. With this, our website has already been created, and several files will be formed within our folder containing the basics of our web app.

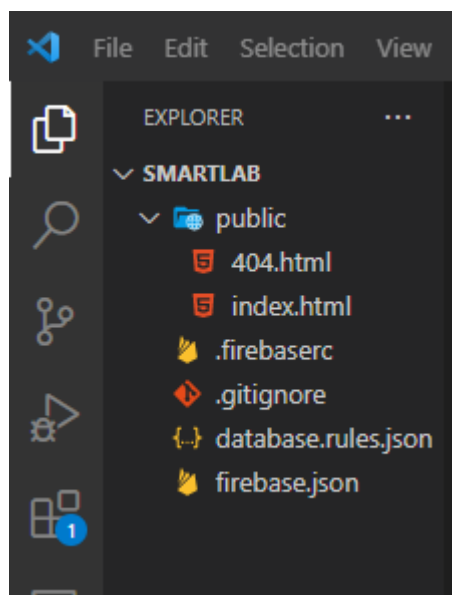


Fig. 35: Firebase VSC Installation part VI

The index.html file will help us to build the web page. It contains the whole structure of how our website looks and the elements that constitute it. The following steps will show the code used in HTML to create the dashboard.

10. We use the terminal command "firebase deploy" to view our web page. With this, our Web App will be visible, and any changes we make to the HTML file or add to our page will be.

```
PS C:\Users\Daniel\Desktop\Machine Learning\Smartlab> firebase deploy

=== Deploying to 'test-b208a'...

i deploying database, hosting
i database: checking rules syntax...
+ database: rules syntax for database test-b208a-default-rtdb is valid
i hosting[test-b208a]: beginning deploy...
i hosting[test-b208a]: found 2 files in public
+ hosting[test-b208a]: file upload complete
i database: releasing rules...
+ database: rules for database test-b208a-default-rtdb released successfully
i hosting[test-b208a]: finalizing version...
+ hosting[test-b208a]: version finalized
i hosting[test-b208a]: releasing new version...
+ hosting[test-b208a]: release complete

+ Deploy complete!

Project Console: https://console.firebase.google.com/project/test-b208a/overview
Hosting URL: https://test-b208a.web.app
```

Fig. 36: Firebase VSC Installation part VII

11. We've completed the process of activating our web app in Firebase. The result should appear as follows.

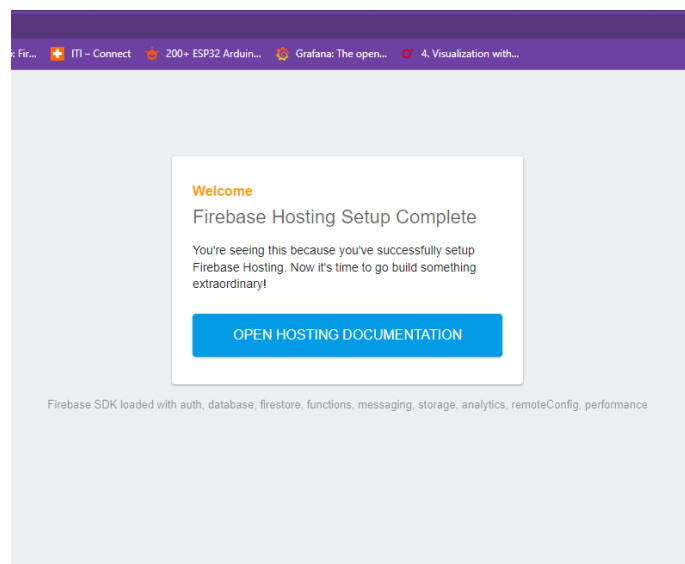


Fig. 37: Firebase VSC Installation part VIII

1.5.5 Web App Dashboard

Once our Web App and the database are activated in real time, we then proceed to create the dashboard for our data website. We will use HTML and JAVA to generate the dashboard type.

Multiple tutorials explain the creation of web pages in HTML and Javascript and definitions such as graphs, tables, and pre-created figures to place them directly on our website. We will use them to create our dashboard.

1. In VS code, we open our Index.html file and create the following code:

```
<html>
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>SMARTLAB Project UL</title>

  <!-- include Firebase SDK -->
  <script src="https://www.gstatic.com/firebasejs/8.8.1/firebase-app.js"></script>

  <!-- include only the Firebase features as you need -->
  <script src="https://www.gstatic.com/firebasejs/8.8.1/firebase-auth.js"></script>
  <script src="https://www.gstatic.com/firebasejs/8.8.1/firebase-database.js"></script>

  <script>
    // Replace with your app config object
    const firebaseConfig = {
      apiKey: "AlzaSyANbD-63734Mde4wGdyFrZeZMkl6HLsqxU",
      authDomain: "SMARTLAB-54c95.firebaseio.com",
      databaseURL: "https://SMARTLAB-54c95.firebaseio.com",
      projectId: "SMARTLAB-54c95",
      storageBucket: "SMARTLAB-54c95.firebaseio.com",
      messagingSenderId: "222497148058",
      appId: "1:222497148058:web:727396592ece3c94042323"
    };

    // Initialise firebase
    firebase.initializeApp(firebaseConfig);

    // Make auth and database references
    const auth = firebase.auth();
    const db = firebase.database();

  </script>

  <!-- include highchartsjs to build the charts-->
  <script src="https://code.highcharts.com/highcharts.js"></script>
  <!-- include to use jquery-->
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
  <!-- include icons from fontawesome-->
  <link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.7.2/css/all.css"
  integrity="sha384-
```

```

fNmOCqbTlWlj8LyTjo7mOUSTjsKC4pOpQbqyi7RrhN7udigRwhKkMHpvLbHG9Sr"
crossorigin="anonymous">
  <!-- include Gauges Javascript library-->
  <script src="https://cdn.rawgit.com/Mikhus/canvas-gauges/gh-
pages/download/2.1.7/all/gauge.min.js"></script>
  <!--reference for favicon-->
  <link rel="icon" type="image/png" href="favicon.png">
  <!--reference a stylesheet-->
  <link rel="stylesheet" type="text/css" href="style.css">

</head>

<body>

  <!--TOP BAR-->
  <div class="topnav">
    <h1>SMARTLAB UL <i class="fas fa-clipboard-list"></i></h1>
  </div>

  <!--AUTHENTICATION BAR (USER DETAILS/LOGOUT BUTTON)-->
  <div id="authentication-bar" style="display: none;" >
    <p><span id="authentication-status">User logged in</span>
    <span id="user-details">USEREMAIL</span>
    <a href="/" id="logout-link">(logout)</a>
  </p>
  </div>

  <!--LOGIN FORM-->
  <form id="login-form" style="display: none;" >
    <div class="form-elements-container">
      <label for="input-email"><b>Email</b></label>
      <input type="text" placeholder="Enter Username" id="input-email" required>

      <label for="input-password"><b>Password</b></label>
      <input type="password" placeholder="Enter Password" id="input-password" required>

      <button type="submit" id="login-button">Login</button>
      <p id="error-message" style="color:red;" ></p>
    </div>
  </form>

  <!--CONTENT (SENSOR READINGS)-->
  <div class="content-sign-in" id="content-sign-in" style="display: none;" >

    <!--LAST UPDATE-->
    <p><span class="date-time">Last update: <span id="lastUpdate"></span></span></p>
    <p>
      Cards: <input type="checkbox" id="cards-checkbox" name="cards-checkbox" checked>

```

```

    Gauges: <input type="checkbox" id="gauges-checkbox" name="gauges-checkbox"
checked>
    Charts: <input type="checkbox" id="charts-checkbox" name="charts-checkbox"
unchecked>
  </p>
  <div id="cards-div">
    <div class="cards">
      <!--TEMPERATURE-->
      <div class="card">
        <p><i class="fas fa-thermometer-half" style="color:#059e8a;" ></i>
TEMPERATURE</p>
        <p><span class="reading"><span id="temp"></span> &deg; C</span></p>
      </div>
      <!--HUMIDITY-->
      <div class="card">
        <p><i class="fas fa-tint" style="color:#00add6;" ></i> HUMIDITY</p>
        <p><span class="reading"><span id="hum"></span> &percent; </span></p>
      </div>

    </div>
  </div>
  <!--GAUGES-->
  <div id="gauges-div">
    <div class="cards">
      <!--TEMPERATURE-->
      <div class="card">
        <canvas id="gauge-temperature"></canvas>
      </div>
      <!--HUMIDITY-->
      <div class="card">
        <canvas id="gauge-humidity"></canvas>
      </div>
    </div>
  </div>
  <!--CHARTS-->
  <div id="charts-div" style="display:none">
    <!--SET NUMBER OF READINGS INPUT FIELD-->
    <div>
      <p> Number of readings: <input type="number" id="charts-range"></p>
    </div>
    <!--TEMPERATURE-CHART-->
    <div class="cards">
      <div class="card">
        <p><i class="fas fa-thermometer-half" style="color:#059e8a;" ></i> TEMPERATURE
CHART</p>
        <div id="chart-temperature" class="chart-container"></div>
      </div>

```

```

</div>
<!--HUMIDITY-CHART-->
<div class="cards">
  <div class="card">
    <p><i class="fas fa-tint" style="color:#00add6;" ></i> HUMIDITY CHART</p>
    <div id="chart-humidity" class="chart-container"></div>
  </div>
</div>

</div>

<!--BUTTONS TO HANDLE DATA-->
<p>
  <!--View data button-->
  <button id="view-data-button">View all data</button>
  <!--Hide data button-->
  <button id="hide-data-button" style="display:none;" >Hide data</button>
  <!--Delete data button-->
  <button id="delete-button" class="deletebtn">Delete data</button>
</p>
<!--Modal to delete data-->
<div id="delete-modal" class="modal" style="display:none">
  <span onclick = "document. getElementById('delete-modal'). style. display='none'"
class="close" title="Close Modal">x</span>
  <form id= "delete-data-form" class="modal-content" action="/">
    <div class="container">
      <h1>Delete Data</h1>
      <p>Are you sure you want to delete all data from database? </p>
      <div class="clearfix">
        <button type="button" onclick="document. getElementById('delete-modal'). style.
display='none'" class="cancelbtn">Cancel</button>
        <button type="submit" onclick="document. getElementById('delete-modal'). style.
display='none'" class="deletebtn">Delete</button>
      </div>
    </div>
  </form>
</div>

<!--TABLE WITH ALL DATA-->
<div class="cards">
  <div class="card" id="table-container" style="display:none;" >
    <table id="readings-table">
      <tr id="thead">
        <th>Timestamp</th>
        <th>Temp (°C)</th>
        <th>Hum (%)</th>
        <th>Pres (hPa)</th>
      </tr>

```

```

        <tbody id="tbody">
        </tbody>
    </table>
    <p><button id="load-data" style="display:none;" >More results... </button></p>
</div>
</div>

<!--INCLUDE JS FILES-->
<script src="scripts/auth.js"></script>
<script src="scripts/charts-definition.js"></script>
<script src="scripts/gauges-definition.js"></script>
<script src="scripts/index.js"></script>

</body>

</html>

```

2. In our public folder, we will create a file called style.css and put the following code.

```

html {
  font-family: Verdana, Geneva, Tahoma, sans-serif;
  display: inline-block;
  text-align: center;
}

body {
  margin: 0;
  width: 100%;
}

.topnav {
  overflow: hidden;
  background-color: #049faa;
  color: white;
  font-size: 1rem;
  padding: 5px;
}

#authentication-bar{
  background-color:mintcream;
  padding-top: 10px;
  padding-bottom: 10px;
}

#user-details{
  color: cadetblue;
}

```

```
}

.content {
  padding: 20px;
}

.card {
  background-color: white;
  box-shadow: 2px 2px 12px 1px rgba(140,140,140,.5);
  padding: 5%;
}

.cards {
  max-width: 800px;
  margin: 0 auto;
  margin-bottom: 10px;
  display: grid;
  grid-gap: 2rem;
  grid-template-columns: repeat(auto-fit, minmax(200px, 2fr));
}

.reading {
  color: #193036;
}

.date-time {
  font-size: 0.8rem;
  color: #1282A2;
}

button {
  background-color: #049faa;
  color: white;
  padding: 14px 20px;
  margin: 8px 0;
  border: none;
  cursor: pointer;
  border-radius: 4px;
}

button:hover {
  opacity: 0.8;
}

.deletebtn {
  background-color: #c52c2c;
}

.form-elements-container {
  padding: 16px;
```



```
width: 250px;
margin: 0 auto;
}

input[type=text], input[type=password] {
  width: 100%;
  padding: 12px 20px;
  margin: 8px 0;
  display: inline-block;
  border: 1px solid #ccc;
  box-sizing: border-box;
}

table {
  width: 100%;
  text-align: center;
  font-size: 0.8rem;
}

tr, td {
  padding: 0.25rem;
}

tr:nth-child(even) {
  background-color: #f2f2f2;
}

tr:hover {
  background-color: #ddd;
}

th {
  position: sticky;
  top: 0;
  background-color: #50b8b4;
  color: white;
}

/* The Modal (background) */
.modal {
  display: none; /* Hidden by default */
  position: fixed; /* Stay in place */
  z-index: 1; /* Sit on top */
  left: 0;
  top: 0;
  width: 100%; /* Full width */
  height: 100%; /* Full height */
  overflow: auto; /* Enable scroll if needed */
  background-color: #474e5d;
  padding-top: 50px;
}
```

```
/* Modal Content/Box */
.modal-content {
  background-color: #fefefe;
  margin: 5% auto 15% auto; /* 5% from the top, 15% from the bottom and centered */
  border: 1px solid #888;
  width: 80%; /* Could be more or less, depending on screen size */
}

/* Style the horizontal ruler */
hr {
  border: 1px solid #f1f1f1;
  margin-bottom: 25px;
}

/* The Modal Close Button (x) */
.close {
  position: absolute;
  right: 35px;
  top: 15px;
  font-size: 40px;
  font-weight: bold;
  color: #f1f1f1;
}

.close:hover,
.close:focus {
  color: #f44336;
  cursor: pointer;
}

/* Clear floats */
.clearfix::after {
  content: "";
  clear: both;
  display: table;
}

/* Change styles for cancel button and delete button on extra small screens */
@media screen and (max-width: 300px) {
  .cancelbtn, .deletebtn {
    width: 100%;
  }
}
```

3. Create inside the public folder a folder called scripts, within this folder, we will create four sub-files for JavaScript: auth.js, index.js, charts-definition.js and gauges-definition.js.

4. For index.js, we will have the following code.

```
// convert epochtime to JavaScripts Date object
function epochToJsDate(epochTime){
  return new Date(epochTime*1000);
}

// convert time to human-readable format YYYY/MM/DD HH:MM:SS
function epochToDateTime(epochTime){
  var epochDate = new Date(epochToJsDate(epochTime));
  var dateTime = epochDate. getFullYear() + "/" +
    ("00" + epochDate. getMonth() + 1). slice(-2) + "/" +
    ("00" + epochDate. getDate()). slice(-2) + " " +
    ("00" + epochDate. getHours()). slice(-2) + ":" +
    ("00" + epochDate. getMinutes()). slice(-2) + ":" +
    ("00" + epochDate. getSeconds()). slice(-2);

  return dateTime;
}

// function to plot values on charts
function plotValues(chart, timestamp, value){
  var x = epochToJsDate(timestamp). getTime();
  var y = Number (value);
  if(chart. series[0]. data. length > 40) {
    chart. series[0]. addPoint([x, y], true, true, true);
  } else {
    chart. series[0]. addPoint([x, y], true, false, true);
  }
}

// DOM elements
const loginElement = document. querySelector('#login-form');
const contentElement = document. querySelector("#content-sign-in");
const userDetailsElement = document. querySelector('#user-details');
const authBarElement = document. querySelector('#authentication-bar');
const deleteButtonElement = document. getElementById('delete-button');
const deleteModalElement = document. getElementById('delete-modal');
const deleteDataFormElement = document. querySelector('#delete-data-form');
const viewDataButtonElement = document. getElementById('view-data-button');
const hideDataButtonElement = document. getElementById('hide-data-button');
const tableContainerElement = document. querySelector('#table-container');
const chartsRangeInputElement = document. getElementById('charts-range');
const loadDataButtonElement = document. getElementById('load-data');
const cardsCheckboxElement = document. querySelector('input[name=cards-checkbox]');
const gaugesCheckboxElement = document. querySelector('input[name=gauges-
checkbox]');
const chartsCheckboxElement = document. querySelector('input[name=charts-checkbox]');
```

```
// DOM elements for sensor readings
const cardsReadingsElement = document. querySelector("#cards-div");
const gaugesReadingsElement = document. querySelector("#gauges-div");
const chartsDivElement = document. querySelector("#charts-div");
const tempElement = document. getElementByld("temp");
const humElement = document. getElementByld("hum");
const updateElement = document. getElementByld("lastUpdate")

// MANAGE LOGIN/LOGOUT UI
const setupUI = (user) => {
  if (user) {
    //toggle UI elements
    loginElement. style. display = 'none';
    contentElement. style. display = 'block';
    authBarElement. style. display = 'block';
    userDetailsElement. style. display = 'block';
    userDetailsElement. innerHTML = user. email;

    // get user UID to get data from database
    var uid = user. uid;
    console. log(uid);

    // Database paths (with user UID)
    var dbPath = 'UsersData/' + uid. toString() + '/readings';
    var chartPath = 'UsersData/' + uid. toString() + '/charts/range';

    //database references
    var dbRef = firebase. database(). ref(dbPath);
    var chartRef = firebase. database(). ref(chartPath);

    // CHARTS
    //number of readings to plot on charts
var chartRange = 40;
    // Get number of readings to plot saved on database (runs when the page first loads and
    whenever there's a change in the database)
    chartRef. on('value', snapshot =>{
      chartRange = Number(snapshot. val());
      console. log(chartRange);
      // Delete all data from charts to update with new values when a new range is selected
      chartT. destroy();
      chartH. destroy();

      // Render new charts to display new range of data
      chartT = createTemperatureChart();
      chartH = createHumidityChart();
      // Update the charts with the new range
```

```
// Get the latest readings and plot them on charts (the number of plotted readings
corresponds to the chartRange value)
dbRef. orderByKey(). limitToLast(chartRange). on('child_added', snapshot =>{
  var jsonData = snapshot. toJSON(); // example: {temperature: 25.02, humidity: 50.20,
timestamp:1641317355}
  // Save values on variables
  var temperature = jsonData. temperature;
  var humidity = jsonData. humidity;
  var timestamp = jsonData. timestamp;
  // Plot the values on the charts
  plotValues(chartT, timestamp, temperature);
  plotValues(chartH, timestamp, humidity);
});

// Update database with new range (input field)
chartsRangeInputElement. onchange = () =>{
  chartRef. set(chartsRangeInputElement. value);
};

//CHECKBOXES
// Checkbox (cards for sensor readings)
cardsCheckboxElement. addEventListener('change', (e) =>{
  if (cardsCheckboxElement. checked) {
    cardsReadingsElement. style. display = 'block';
  }
  else{
    cardsReadingsElement. style. display = 'none';
  }
});

// Checkbox (gauges for sensor readings)
gaugesCheckboxElement. addEventListener('change', (e) =>{
  if (gaugesCheckboxElement. checked) {
    gaugesReadingsElement. style. display = 'block';
  }
  else{
    gaugesReadingsElement. style. display = 'none';
  }
});

// Checkbox (charta for sensor readings)
chartsCheckboxElement. addEventListener('change', (e) =>{
  if (chartsCheckboxElement. checked) {
    chartsDivElement. style. display = 'block';
  }
  else{
    chartsDivElement. style. display = 'none';
  }
});
```

```
// CARDS
// Get the latest readings and display on cards
dbRef. orderByKey(). limitToLast(1). on('child_added', snapshot =>{
  var jsonData = snapshot. toJSON(); // example: {temperature: 25.02, humidity: 50.20,
timestamp:1641317355}
  var temperature = jsonData. temperature;
  var humidity = jsonData. humidity;
  var timestamp = jsonData. timestamp;
  // Update DOM elements
  tempElement. innerHTML = temperature;
  humElement. innerHTML = humidity;
  updateElement. innerHTML = epochToDateTime(timestamp);
});

// GAUGES
// Get the latest readings and display on gauges
dbRef. orderByKey(). limitToLast(1). on('child_added', snapshot =>{
  var jsonData = snapshot. toJSON(); // example: {temperature: 25.02, humidity: 50.20,
timestamp:1641317355}
  var temperature = jsonData. temperature;
  var humidity = jsonData. humidity;
  var timestamp = jsonData. timestamp;
  // Update DOM elements
var gaugeT = createTemperatureGauge();
var gaugeH = createHumidityGauge();
  gaugeT. draw();
  gaugeH. draw();
  gaugeT. value = temperature;
  gaugeH. value = humidity;
  updateElement. innerHTML = epochToDateTime(timestamp);
});

// DELETE DATA
// Add event listener to open modal when click on "Delete Data" button
deleteButtonElement. addEventListener('click', e =>{
  console. log("Remove data");
and. preventDefault;
  deleteModalElement. style. display="block";
});

// Add event listener when delete form is submitted
deleteDataFormElement. addEventListener('submit', (e) => {
  // delete data (readings)
  dbRef. remove();
});

// TABLE
```

```

var lastReadingTimestamp; //saves last timestamp displayed on the table
//function that creates the table with the first 100 readings
function createTable(){
  // append all data to the table
  var firstRun = true;
  dbRef. orderByKey(). limitToLast(100). on('child_added', function(snapshot) {
    if (snapshot. exists()) {
      var jsonData = snapshot. toJSON();
      console. log(jsonData);
      var temperature = jsonData. temperature;
      var humidity = jsonData. humidity;
      var timestamp = jsonData. timestamp;
      var content = "";
      content += '<tr>';
      content += '<td>' + epochToDateTime(timestamp) + '</td>';
      content += '<td>' + temperature + '</td>';
      content += '<td>' + humidity + '</td>';
      content += '</tr>';
      $('#tbody'). prepend(content);
      // Save lastReadingTimestamp --> corresponds to the first timestamp on the returned
      snapshot data
      if (firstRun){
        lastReadingTimestamp = timestamp;
        firstRun=false;
        console. log(lastReadingTimestamp);
      }
    }
  });
};

// append readings to table (after pressing More results... button)
function appendToTable(){
  var dataList = []; // saves list of readings returned by the snapshot (oldest-->newest)
  var reversedList = []; // the same as previous, but reversed (newest--> oldest)
  console. log("APEND");
  dbRef. orderByKey(). limitToLast(100). endAt(lastReadingTimestamp). once('value',
function(snapshot) {
  // convert the snapshot to JSON
  if (snapshot. exists()) {
    snapshot. forEach(element => {
var jsonData = element. toJSON();
    dataList. push(jsonData); // create a list with all data
  });
    lastReadingTimestamp = dataList[0]. timestamp; //oldest timestamp corresponds to
the first on the list (oldest --> newest)
    reversedList = dataList. reverse(); // reverse the order of the list (newest data -->
oldest data)
  }
});
}

```

```
var firstTime = true;
// loop through all elements of the list and append to table (newest elements first)
reversedList.forEach(element =>{
  if (firstTime){ // ignore first reading (it's already on the table from the previous query)
    firstTime = false;
  }
  else{
    var temperature = element. temperature;
    var humidity = element. humidity;
    var timestamp = element. timestamp;
    var content = "";
    content += '<tr>';
    content += '<td>' + epochToDateTime(timestamp) + '</td>';
    content += '<td>' + temperature + '</td>';
    content += '<td>' + humidity + '</td>';
    content += '</tr>';
    $('#tbody'). append(content);
  }
});
}
});
}

viewDataButtonElement. addEventListener('click', (e) =>{
  // Toggle DOM elements
  tableContainerElement. style. display = 'block';
  viewDataButtonElement. style. display = 'none';
  hideDataButtonElement. style. display = 'inline-block';
  loadDataButtonElement. style. display = 'inline-block'
  createTable();
});

loadDataButtonElement. addEventListener('click', (e) => {
  appendToTable();
});

hideDataButtonElement. addEventListener('click', (e) => {
  tableContainerElement. style. display = 'none';
  viewDataButtonElement. style. display = 'inline-block';
  hideDataButtonElement. style. display = 'none';
});

// IF USER IS LOGGED OUT
} else{
  // toggle UI elements
  loginElement. style. display = 'block';
  authBarElement. style. display = 'none';
  userDetailsElement. style. display = 'none';
```



```
contentElement. style. display = 'none';
}
]
```

5. For auth.js we will have the following code.

```
document. addEventListener("DOMContentLoaded", function(){
  // listen for auth status changes
  auth. onAuthStateChanged(user => {
    if (user) {
      console. log("user logged in");
      console. log(user);
      setupUI(user);
      var uid = user. uid;
      console. log(uid);
    } else {
      console. log("user logged out");
      setupUI();
    }
  });

  login
  const loginForm = document. querySelector('#login-form');
  loginForm. addEventListener('submit', (e) => {
    e. preventDefault();
    // get user info
    const email = loginForm['input-email']. value;
    const password = loginForm['input-password']. value;
    // log the user in
    auth. signInWithEmailAndPassword(email, password). then((cred) => {
      // close the login modal & reset form
      loginForm. reset();
      console. log(email);
    })
    . catch((error) =>{
      const errorCode = error. code;
      const errorMessage = error. message;
      document. getElementById("error-message"). innerHTML = errorMessage;
      console. log(errorMessage);
    });
  });

  Logout
  const logout = document. querySelector('#logout-link');
  logout. addEventListener('click', (e) => {
    e. preventDefault();
    auth. signOut();
  });
});
```

```
});
```

6. For charts-definition.js, we will have the following code.

```
// Create the charts when the web page loads
window.addEventListener('load', onload);

function onload(event){
  chartT = createTemperatureChart();
  chartH = createHumidityChart();
}

// Create Temperature Chart
function createTemperatureChart() {
  var chart = new Highcharts.Chart({
    chart: {
      renderTo: 'chart-temperature',
      type: 'spline'
    },
    series: [
      {
        name: 'DHT11'
      }
    ],
    title: {
      text: undefined
    },
    plotOptions: {
      line: {
        animation: false,
        dataLabels: {
          enabled: true
        }
      }
    },
    xAxis: {
      type: 'datetime',
      dateTimeLabelFormats: { second: '%H:%M:%S' }
    },
    yAxis: {
      title: {
        text: 'Temperature Celsius Degrees'
      }
    },
    credits: {
      enabled: false
    }
  });
}
```

```
return chart;
}

// Create Humidity Chart
function createHumidityChart(){
  var chart = new Highcharts. Chart({
    chart:{
      renderTo:'chart-humidity',
      type: 'spline'
    },
    series: [{
      name: 'DHT11'
    }],
    title: {
      text: undefined
    },
    plotOptions: {
      line: {
        animation: false,
        dataLabels: {
          enabled: true
        }
      },
    },
    series: {
      color: '#50b8b4'
    }
  },
  xAxis: {
    type: 'datetime',
    dateTimeLabelFormats: [ second: '%H:%M:%S' ]
  },
  yAxis: {
    title: {
      text: 'Humidity (%)'
    }
  },
  credits: {
    enabled: false
  }
});
return chart;
}
```

7. For gauges-definition.js, the following code.

```
// Create Temperature Gauge
function createTemperatureGauge() {
  var gauge = new LinearGauge({
```

```
renderTo: 'gauge-temperature',
width: 120,
height: 400,
units: "Temperature C",
minValue: 0,
startAngle: 90,
ticksAngle: 180,
maxValue: 40,
colorValueBoxRect: "#049faa",
colorValueBoxRectEnd: "#049faa",
colorValueBoxBackground: "#f1fbfc",
valueDec: 2,
valueInt: 2,
majorTicks: [
  "0",
  "5",
  "10",
  "15",
  "20",
  "25",
  "30",
  "35",
  "40"
],
minorTicks: 4,
strokeTicks: true,
highlights: [
  {
    "from": 30,
    "to": 40,
    "color": "rgba(200, 50, 50, .75)"
  }
],
colorPlate: "#fff",
colorBarProgress: "#CC2936",
colorBarProgressEnd: "#049faa",
borderShadowWidth: 0,
borders: false,
needleType: "arrow",
needleWidth: 2,
needleCircleSize: 7,
needleCircleOuter: true,
needleCircleInner: false,
animationDuration: 1500,
animationRule: "linear",
barWidth: 10,
});
return gauge;
```

```
}

// Create Humidity Gauge
function createHumidityGauge()
{
    var gauge = new RadialGauge({
        renderTo: 'gauge-humidity',
        width: 300,
        height: 300,
        units: "Humidity (%)",
        minValue: 0,
        maxValue: 100,
        colorValueBoxRect: "#049faa",
        colorValueBoxRectEnd: "#049faa",
        colorValueBoxBackground: "#f1fbfc",
        valueInt: 2,
        majorTicks: [
            "0",
            "20",
            "40",
            "60",
            "80",
            "100"
        ],
        minorTicks: 4,
        strokeTicks: true,
        highlights: [
            {
                "from": 80,
                "to": 100,
                "color": "#03CoC1"
            }
        ],
        colorPlate: "#fff",
        borderShadowWidth: 0,
        borders: false,
        needleType: "line",
        colorNeedle: "#007F80",
        colorNeedleEnd: "#007F80",
        needleWidth: 2,
        needleCircleSize: 3,
        colorNeedleCircleOuter: "#007F80",
        needleCircleOuter: true,
        needleCircleInner: false,
        animationDuration: 1500,
        animationRule: "linear"
    });
    return gauge;
}
```

With this, we have already finished the creation of our dashboard. As mentioned above, we use some elements already manufactured and created by the community that uses this software.

The result should look like the following:

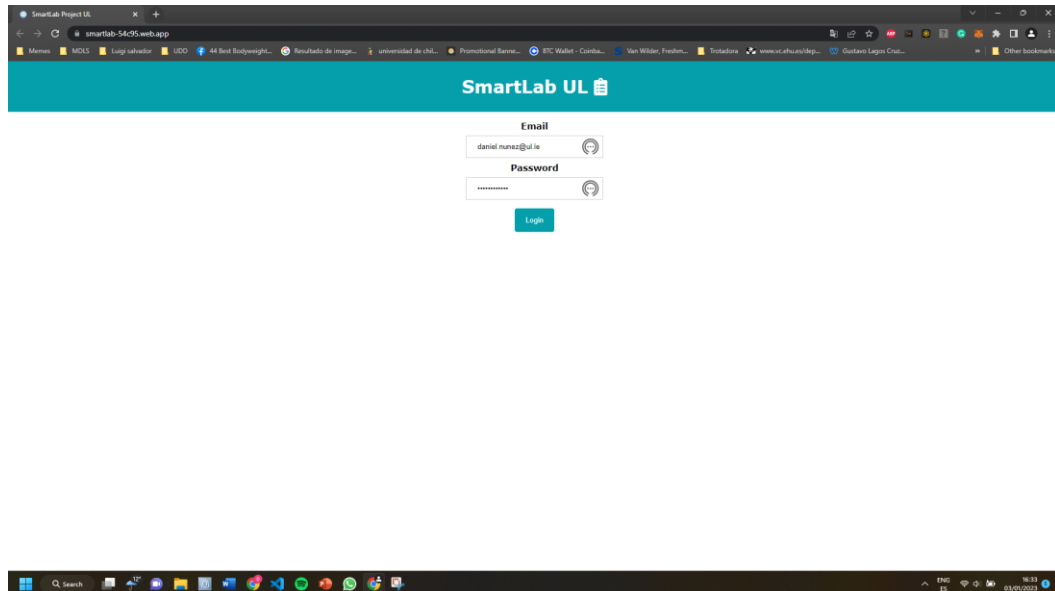


Fig. 38: Firebase Front Authentication panel

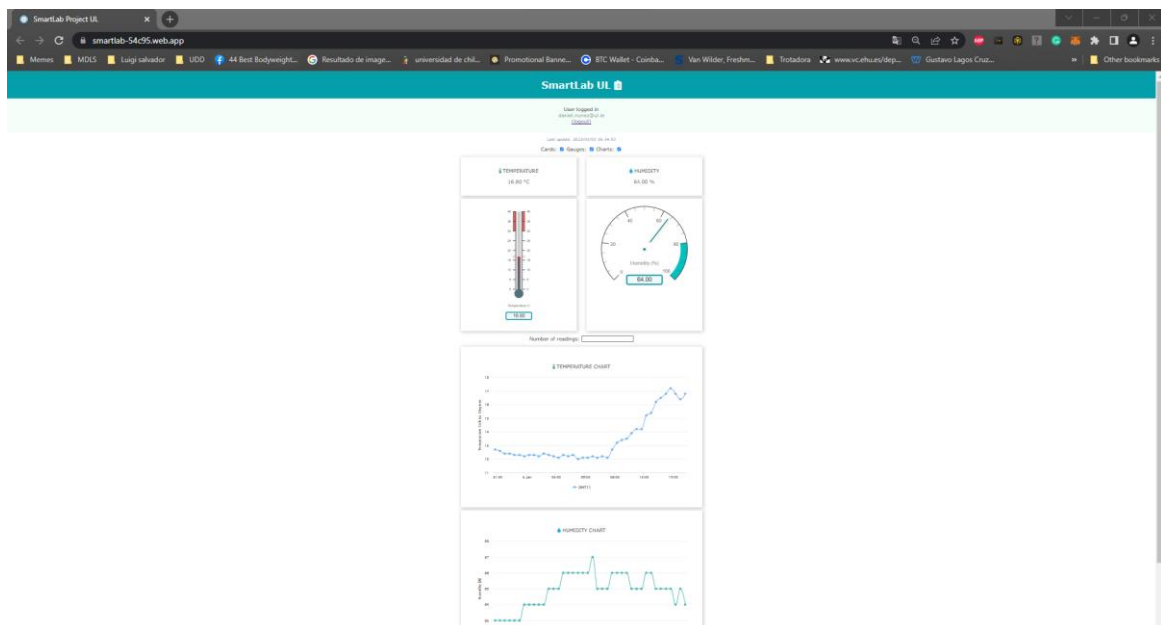


Fig. 39: Firebase dashboard panel

1.5.6 CAD model

Autodesk's Fusion 360 software generated 3D and DXF models for laser cutting. For this version of the DIY sensor, 3D modelling, and the generation of CAD documents were used for the creation of the case that would cover the sensor. Models generated by the open-source community were used to achieve the correct dimension of the components.

The arrangement of the components for version v1 is as follows:

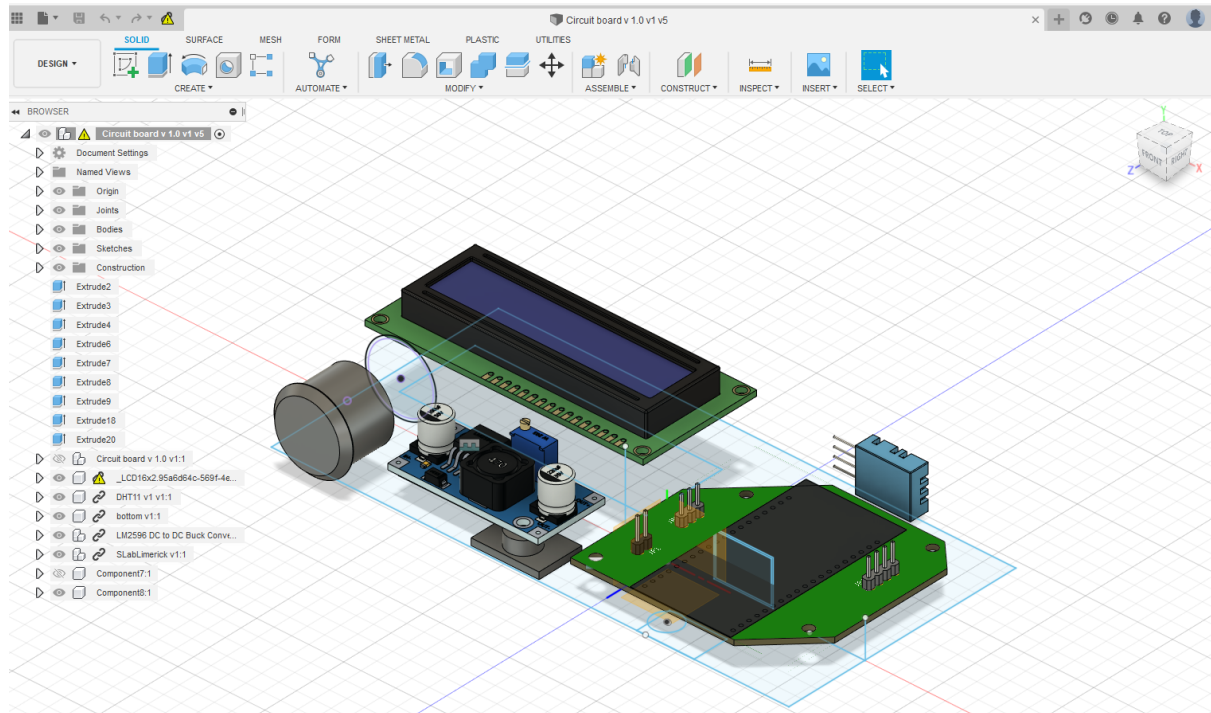


Fig. 40: 3D model DIY sensor V1

With this arrangement, it is possible to obtain the case's interior dimensions that must wrap the components. With these dimensions, a DXF is generated with all the pieces to be cut in laser-cut.

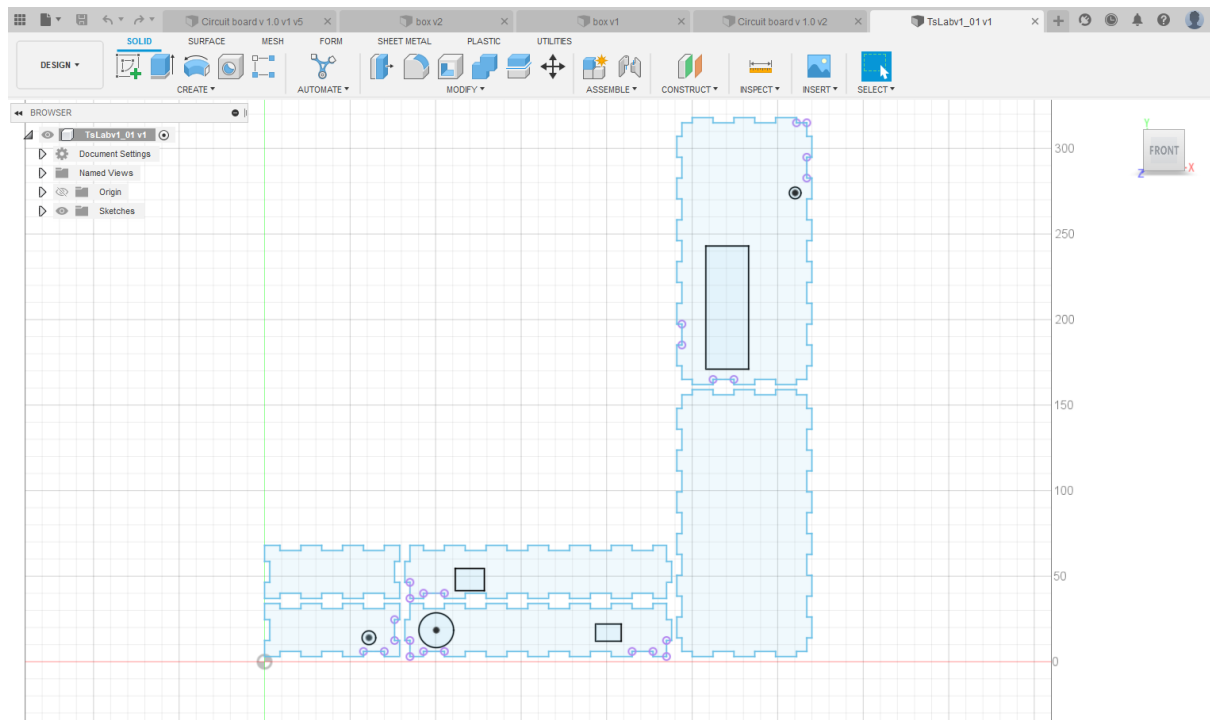


Fig. 41: DXF model DIY sensor V1

In the same way, the models for the compact version are generated and simplified by removing the LCD screen and the DC-DC converter.

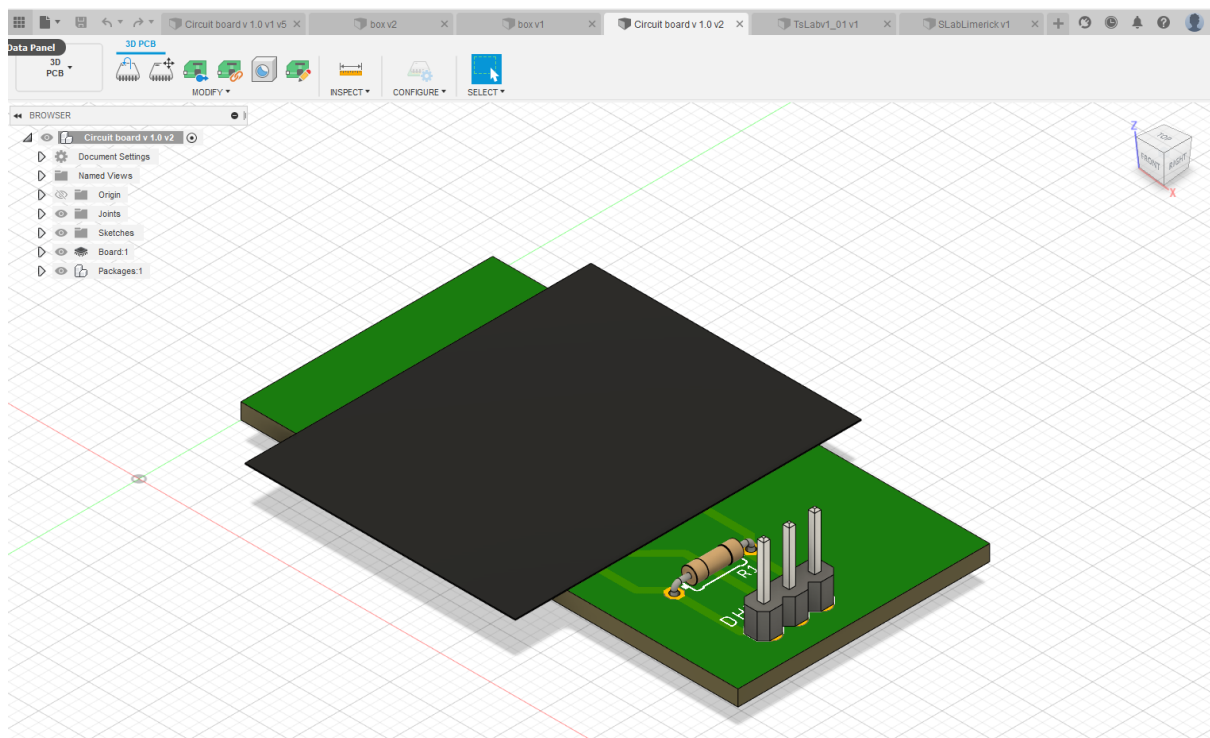


Fig. 42: 3D model DIY sensor V1 compact

DXF model for laser cutting

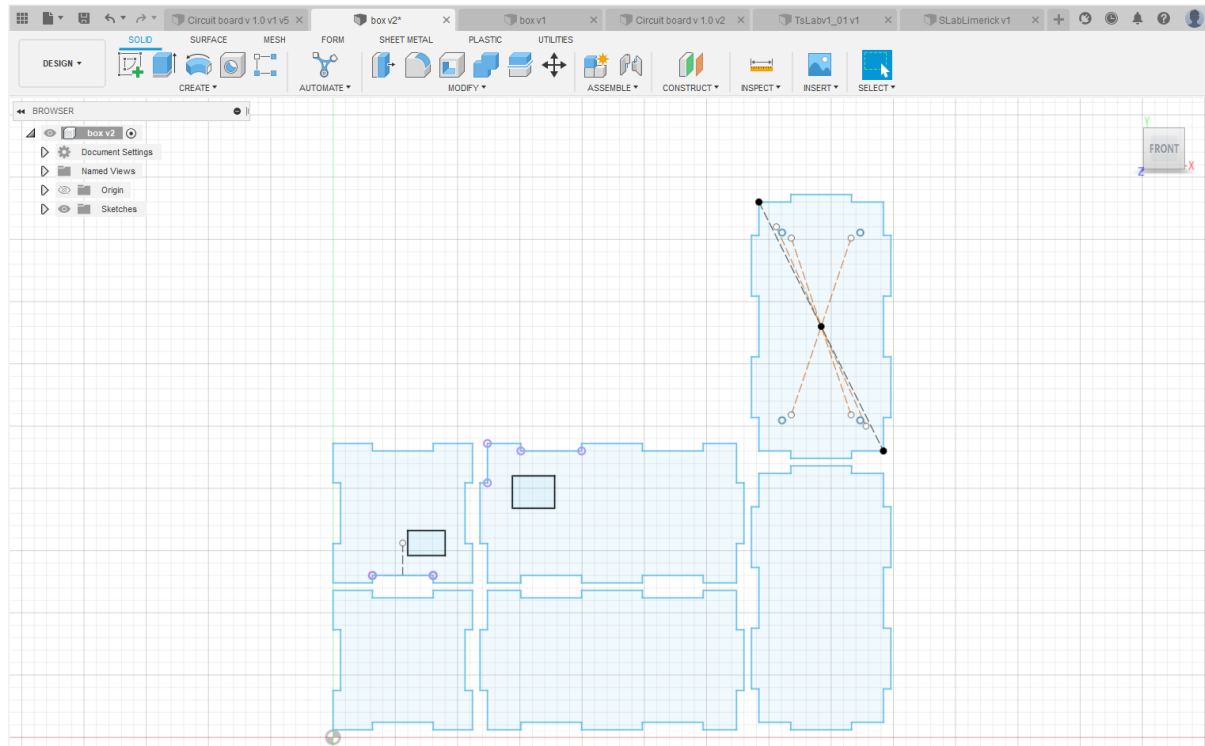


Fig. 43: DXF model DIY sensor V1 compact

For both the standard and compact v1 versions, 3 mm white acrylic is used.

1.6 DIY Sensor v2 process

This version contains more elements than previous versions. Section 3.2 explains that the sensor is intended to be an economical version of commercially available Smart Sensors, particularly the Milesight AM300 Series.

The v2 sensor allows you to measure nine environmental variables.

1. Temperature
2. Relative humidity
3. Barometric pressure
4. Particulate matter 1.0, 2.5 and 10.
5. VOCs (volatile organic concentration)
6. eCO₂ (Carbon dioxide and equivalent)

The sensors used are described in Table 1. The microcontroller used for these versions is an ESP32 Firebeetle of the DFrobot brand.

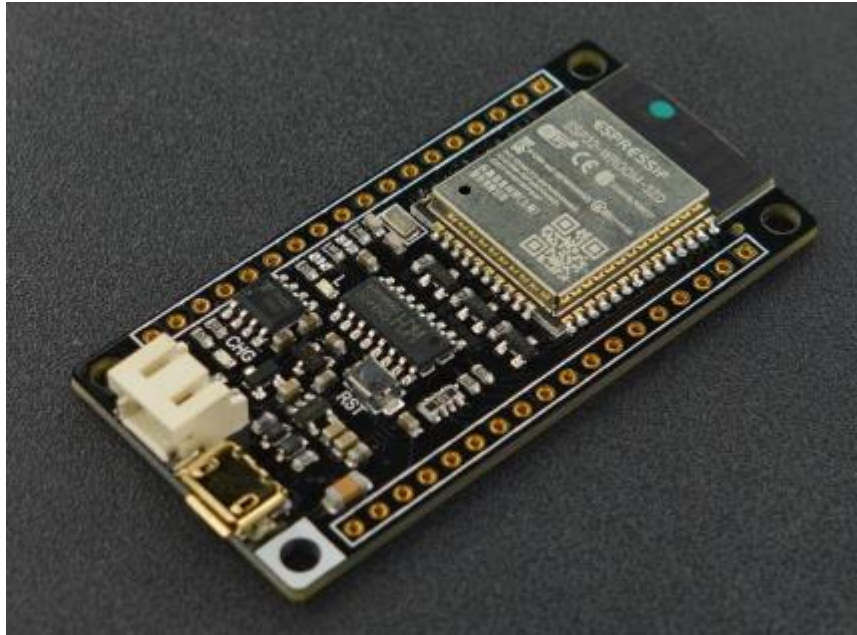


Fig. 44: ESP32 firebeetle V4, DFRobot

The characteristics of this microcontroller are similar to those of the microcontroller used in the DIY sensor v1. The main difference is that it has a socket for the battery and Bluetooth 4.0.

We then explain the construction of this version of the sensor step by step.

4.6.1 Electronic diagram

Autodesk Eagle is used to generate electronic diagrams of the components. The components are simplified because each sensor has its circuit board that simplifies the connection points.

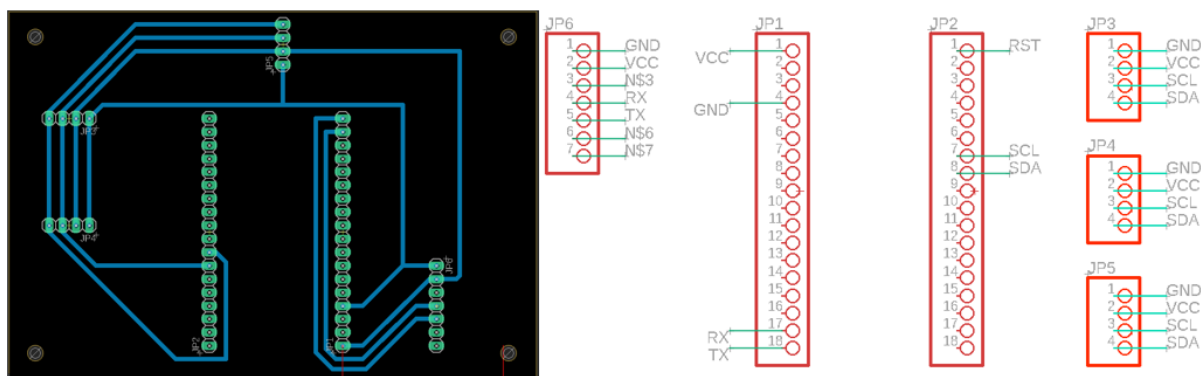


Fig. 45: Electronic diagram DIY sensor v2

Each component has its electronic diagram, and its features can be found in the corresponding datasheets.

4.6.2 Sensor Code

Each sensor manufacturer provides codes, usage suggestions, and calibration functions. The work is to understand each code and adapt it to the use we want to give it. Each of these codes can be found in their respective documentation. Additionally, using a Wi-Fi connection is considered the first version.

On the other hand, for version v2 Firebase was not used as an IoT platform. This is because Firebase requires HH to accommodate various needs. For this version, an IoT platform is developed using the web app already available for free by Cayenne of My devices.

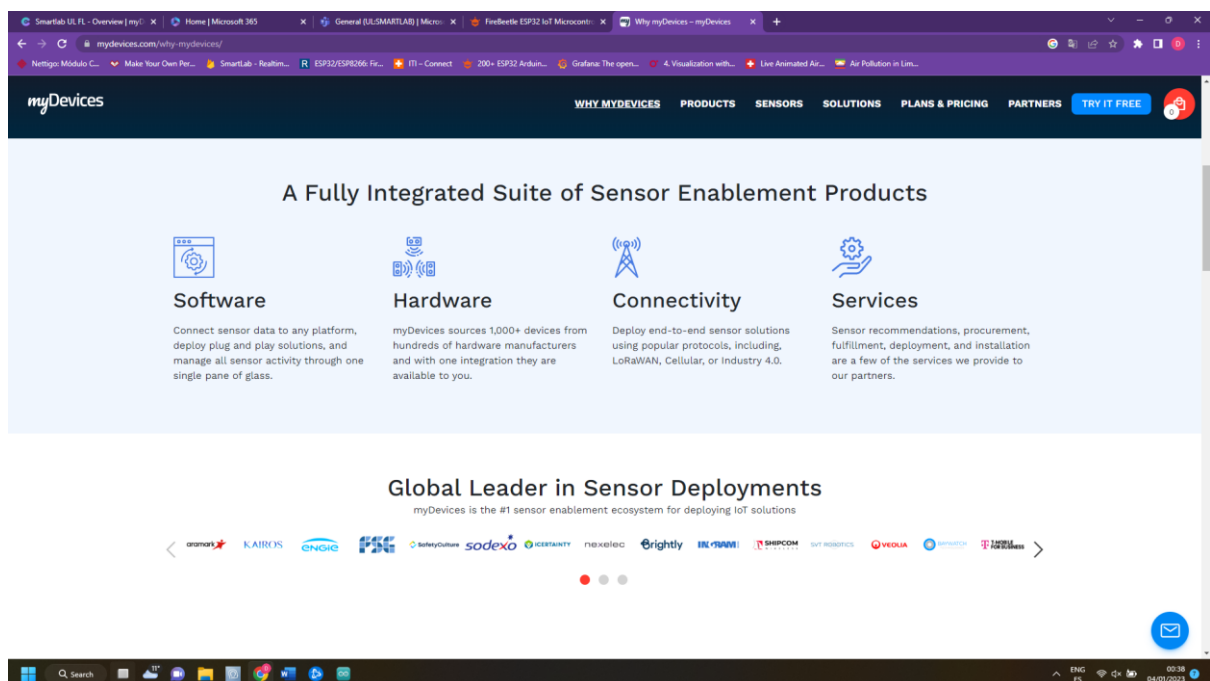


Fig. 46: Mydevices webpage

My device is an IoT platform that integrates different products, in addition to working with many IoT device manufacturers.

Who We Work With



Industry Experts

Specializes in Food Safety, Energy Consulting, Water Management, Agriculture, and more. They provide end customers with specialized solutions by combining their industry expertise with easy integratable sensor data.



System Integrators

Assembles complex technical solutions for end customers, often mixing and matching technologies. Working under specific customer requirements they can easily mix and match sensors to deploy any solution.



Managed Service Providers

Delivers food, facility, safety and other services to their customers. They integrate sensors into their services to enhance operational efficiency to reduce cost and provide exceptional service that outperforms their competition.



Software Solution Providers

Develops, markets, and sells software solutions across a variety of verticals. They integrate sensors to expand their offering adding significant value to their existing solutions.



Carriers and Operators

Provides mobility and connectivity solutions to business customers. They integrate sensors and offer packaged solutions to expand their product and service offerings in IoT to include more B2B SaaS solutions.



Hardware Manufacturers

Manufactures IoT sensors and gateways and want an app that instantly connects their devices to any IoT platform providing their end customers with complete turnkey solutions.

Fig. 47: Mydevices Services

Among its products, the Cayenne platform stands out, which allows us to integrate several IoT devices for free, access data storage and easily customisable and scalable dashboard visualisation. In addition, it highlights a simple integration, which allows users without an advanced level to access their data from their devices.

Additionally, it allows the creation of web apps like the one created in Firebase, but in a professional way. This option is paid, but it will enable you to test and test gateways and personal devices and create a complete dashboard, with the possibility of cross-platform integrations, such as:

- Amazon web services
- Azure
- Slack
- Google Cloud
- MQTT
- Among others.

To connect to Cayenne, we must create an account with <https://developers.mydevices.com/cayenne/features/>

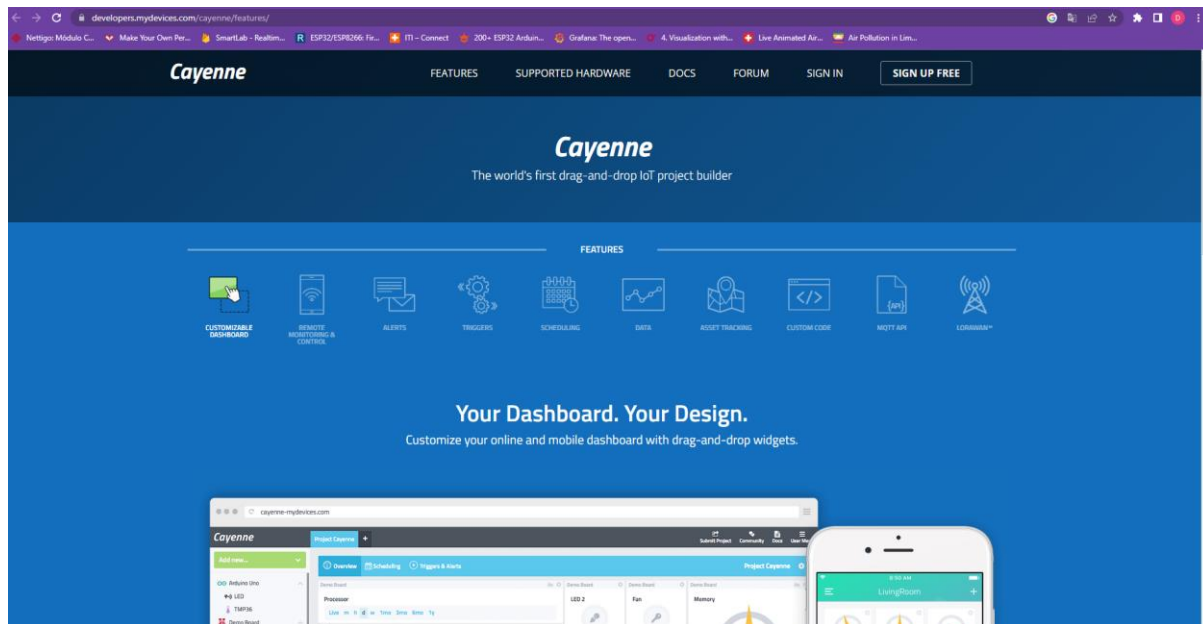


Fig. 48: Cayenne IoT platform

Once our account is created and entered the Cayenne platform, we integrate our DIY devices. For this, the platform allows us to integrate different popular microcontrollers.

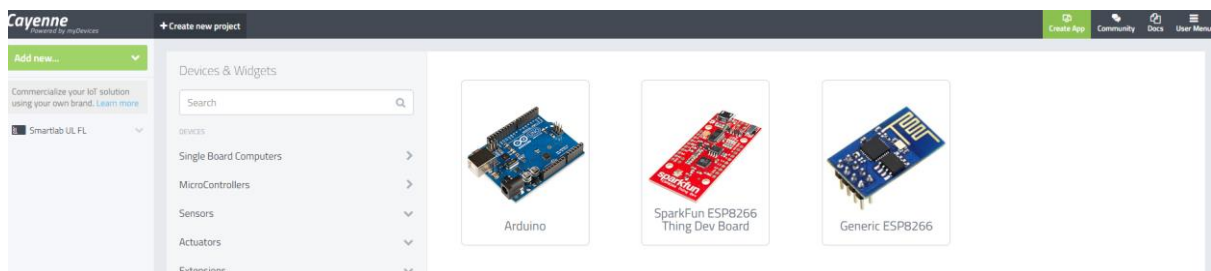


Fig. 49: Cayenne IoT platform part I

By selecting the generic ESP8266, the platform will guide us through adding our DIY sensor and giving us all the access credentials by MQTT.

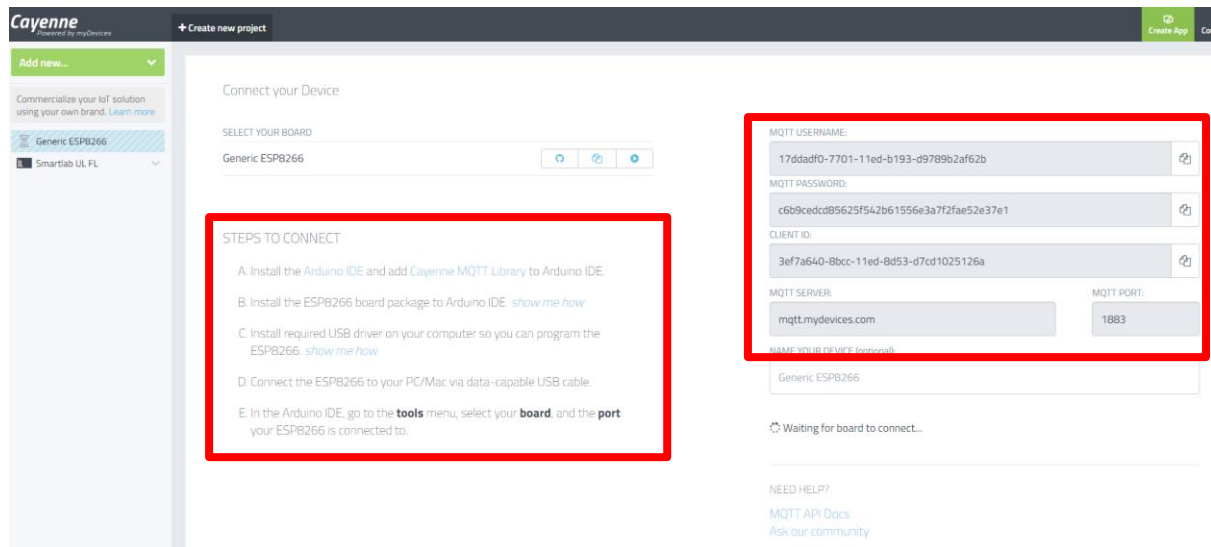


Fig. 50: Cayenne IoT platform part II

We must follow the instructions and add the corresponding codes. Once this process is completed, we proceed to the code generation to integrate our DIY sensor. The DIY v2 code is as follows:

```
//Libraries
#include <Arduino.h>
#include "DFRobot_BME280.h" //BME280 Sensor
#include "sensirion_common.h" //VOC sensor
#include "sgp30.h" // VOC sensor
#include <LCD_I2C.h> //LCD screen
// Cayenne
#define CAYENNE_PRINT Serial
#include <CayenneMQTTESP32.h>

//Wi-Fi network info.
char ssid[] = "SSID";
char wifiPassword[] = "Pass";

// Cayenne authentication info. This should be obtained from the Cayenne Dashboard.
char username[] = "MQTTUser";
char password[] = "MQTTPASS";
char clientID[] = "MQTTID";

//Cayenne

//Screen
LCD_I2C lcd(ox27, 16, 2); // Default address of most PCF8574 modules, change according

//Screen
```

Libraries for the operation of the code and functions of each sensor

Wireless connection credentials

MQTT access credentials to cayenne. Fig 7.2.2.5

```
// PM2.5/5/10 sensor init
#define LENG 31 //0x42 + 31 bytes equal to 32 bytes
unsigned char buf[LENG];

int PM01Value=0; //define PM1.0 value of the air detector module
int PM2_5Value=0; //define PM2.5 value of the air detector module
int PM10Value=0; //define PM10 value of the air detector module

// Finish PM2.5/5/10 sensor finish

// BME280 Sensor init
typedef DFRobot_BME280_IIC BME; // ***** use abbreviations instead of full names
*****

/**IIC address is 0x77 when pin SDO is high */
/**IIC address is 0x76 when pin SDO is low */
BME bme(&Wire, 0x77); // select TwoWire peripheral and set sensor address

#define SEA_LEVEL_PRESSURE 1013.2f

// show last sensor operate status
void printLastOperateStatus(BME::eStatus_t eStatus)
{
switch(eStatus) {
case BME::eStatusOK: Serial.println("everything ok"); break;
case BME::eStatusErr: Serial.println("unknow error"); break;
case BME::eStatusErrDeviceNotDetected: Serial.println("device not detected"); break;
case BME::eStatusErrParameter: Serial.println("parameter error"); break;
default: Serial.println("unknow status"); break;
}
}

// BME280 Sensor finish

//VOC sensor init
//VOC sensor finish

void setup()
{
//PM2.5/5/10 sensor init
Serial.begin(9600); //use serial0
Serial.setTimeout(1500); //set the Timeout to 1500ms, longer than the data
transmission periodic time of the sensor
// Finish PM2.5/5/10 sensor finish
Cayenne.begin(username, password, clientID, ssid, wifiPassword);

// BME280 Sensor init
Serial.println("bme read data test");
while(bme.begin() != BME::eStatusOK) {
```

Initial configuration for particulate matter sensor

BME280 Sensor Configuration
Sea level pressure is set as reference

Start the system connected to the Cayenne platform

```

Serial. println("bme begin faild");
printLastOperateStatus(bme. lastOperateStatus);
delay(2000);
}
Serial. println("bme begin success");
delay(100);
// BME280 Sensor finish

//VOC sensor
s16 err;
u32 ah = 0;
u16 scaled_ethanol_signal, scaled_h2_signal;
/* Init module,Reset all baseline,The initialisation takes up to around 15 seconds, during
which
all APIs measuring IAQ(Indoor air quality ) output will not change. Default value is
400(ppm) for co2,0(ppb) for tvoc*/
while (sgp_probe() != STATUS_OK) {
    Serial.Println("SGP failed");
    while (1);
}
/*Read H2 and Ethanol signal in the way of blocking*/
err = sgp_measure_signals_blocking_read(&scaled_ethanol_signal,
&scaled_h2_signal);
if (err == STATUS_OK) {
    Serial. println("get ram signal!" );
} else {
    Serial. println("error reading signals");
}

err = sgp_iaq_init();
//

//Screen
lcd. begin();

lcd. backlight();
//Screen

}

void loop()
{
    //Cayenne
    Cayenne. loop();
    //cayenne

```

Initial configuration for VOC and eCO2 sensor


```

}

CAYENNE_OUT_DEFAULT()
{
    //BME280
    float temp = bme. getTemperature();
    uint32_t press = bme. getPressure();
    float alti = bme. calAltitude(SEA_LEVEL_PRESSURE, press);
    float humi = bme. getHumidity();
    //BME280

    //VOC
    float Ah = 216.70*((humi/100)*6.112*exp((17.62*temp)/(243.12+temp)))/(273.15+temp);
    sgp_set_absolute_humidity(Ah);
    s16 err = 0;
    u16 tvoc_ppb, co2_eq_ppm;
    err = sgp_measure_iaq_blocking_read(&tvoc_ppb, &co2_eq_ppm);

    //VOC

    //PM2.5/5/10 sensor init
    if(Serial. find(0x42)){ //start to read when detect 0x42
        Serial. readBytes(buf, LENG);

        if(buf[0] == 0x4d){
            if(checkValue(buf, LENG)){
                PM01Value=transmitPM01(buf); //count PM1.0 value of the air detector module
                PM2_5Value=transmitPM2_5(buf); //count PM2.5 value of the air detector module
                PM10Value=transmitPM10(buf); //count PM10 value of the air detector module
            }
        }
    }

    static unsigned long serialTimer=millis();
    static unsigned long LCDTimer=millis();
    if (millis() - serialTimer >=13000)
    {
        serialTimer=millis();
        LCDTimer=millis();
        Cayenne. virtualWrite(0, temp, TYPE_TEMPERATURE, UNIT_CELSIUS);
        Cayenne. virtualWrite(1, press/100.0, TYPE_BAROMETRIC_PRESSURE,
UNIT_HECTOPASCAL);
        Cayenne.virtualWrite(2,High);
        Cayenne. virtualWrite(3, humi, "Humidity", "%");
        Cayenne. virtualWrite(4, PM01Value);
        Cayenne. virtualWrite(5, PM2_5Value, "Particular Material", "ug/m3");
        Cayenne. virtualWrite(6, PM10Value);
        Cayenne. virtualWrite(7, tvoc_ppb, "Volatile Organic", "Parts per Billion");
    }
}

```

Methods of accessing data from different sensors: BME280, VOC, PM

Correction method for VOCs and eCO2 using absolute humidity.

Clocks are created that allow the measurement of data from time to time. In this case every 13 seconds.

Method for sending data to Cayenne. (element,value)

```

Cayenne. virtualWrite(8, co2_eq_ppm, "Carbon Dioxide", "Parts per Million");

Serial, serial. println();
Serial. println("===== start print =====");
Serial. print("temperature (unit Celsius): "); Serial. println(temp);
Serial. print("pressure (unit hpa): "); Serial. println(press/100.0);
Serial. print("altitude (unit meter): "); Serial. println(alti);
Serial. print("humidity (unit percent): "); Serial. println(humi);

Serial. print("PM1.0: ");
Serial. print(PM01Value);
Serial, serial. println("ug/m3");

Serial. print("PM2.5: ");
Serial. print(PM2_5Value);
Serial, serial. println("ug/m3");

Serial. print("PM10: ");
Serial. print(PM10Value);
Serial, serial. println("ug/m3");

Serial. print("tVOC Concentration:");
Serial. print(tvoc_ppb);
Serial. println("ppb");

Serial. print("CO2eq Concentration:");
Serial. print(co2_eq_ppm);
Serial, serial. println("ppm");

Serial. print("Absolute Humidity:");
Serial. print(Ah);

```

Method for displaying the values of different sensors in serial communication

```

Serial, serial. println();

Serial. println("===== end print =====");
}

lcd. clear();
while (millis() - LCDTimer >= 0 && millis() - LCDTimer <= 2000)
{

lcd. setCursor(0, 0);
lcd. print("Temp(C): "); // You can make spaces using well... spaces
lcd. setCursor(9, 0);
lcd. print(temp); // You can make spaces using well... spaces
lcd. setCursor(0, 1);
lcd. print("Hum(%)"); // You can make spaces using well... spaces
lcd. setCursor(8, 1);

```

```

    lcd. print(humi); // You can make spaces using well... spaces

}

lcd. clear();
while (millis() - LCDTimer > 2000 && millis() - LCDTimer <= 4000)
{

    lcd. setCursor(0, 0);
    lcd. print("P(hPa: "); // You can make spaces using well... spaces
    lcd. setCursor(8, 0);
    lcd. print(press/100.0); // You can make spaces using well... spaces

}

lcd. clear();
while (millis() - LCDTimer > 4000 && millis() - LCDTimer <= 6000)
{

    lcd. setCursor(0, 0);
    lcd. print(" PM 1.0-2.5-10 "); // You can make spaces using well... spaces
    lcd. setCursor(4, 1);
    lcd. print("ug/m3"); // You can make spaces using well... spaces

}

lcd. clear();
while (millis() - LCDTimer > 6000 && millis() - LCDTimer <= 8000)
{

    lcd. setCursor(0, 0);
    lcd. print("1.0: "); // You can make spaces using well... spaces
    lcd. setCursor(5, 0);
    lcd. print(PM01Value); // You can make spaces using well... spaces
    lcd. setCursor(0, 1);
    lcd. print("2.5: "); // You can make spaces using well... spaces
    lcd. setCursor(5, 1);
    lcd. print(PM2_5Value); // You can make spaces using well... spaces

}

lcd. clear();
while (millis() - LCDTimer > 8000 && millis() - LCDTimer <= 10000)
{

    lcd. setCursor(0, 0);
    lcd. print("10: "); // You can make spaces using well... spaces

```

Each While function aims to display a group of sensor values for a fixed amount of time. 2 seconds for each group.

```
    lcd.setCursor(5, 0);
    lcd.print(PM10Value); // You can make spaces using well... spaces
}

lcd.clear();
while (millis() - LCDTimer > 10000 && millis() - LCDTimer <= 12000)
{

    lcd.setCursor(0, 0);
    lcd.print("tVOC(ppb): "); // You can make spaces using well... spaces
    lcd.setCursor(11, 0);
    lcd.print(tvoc_ppb); // You can make spaces using well... spaces
    lcd.setCursor(0, 1);
    lcd.print("CO2(ppm): "); // You can make spaces using well... spaces
    lcd.setCursor(10, 1);
    lcd.print(co2_eq_ppm); // You can make spaces using well... spaces

}

// Finish PM2.5/5/10 sensor finish

}

//PM2.5/5/10 sensor init
char checkValue(unsigned char *thebuf, char leng)
{
    char receiveflag=0;
    int receiveSum=0;

    for(int i=0; i<(leng-2); i++){
        receiveSum=receiveSum+thebuf[i];
    }
    receiveSum=receiveSum + 0x42;

    if(receiveSum == ((thebuf[leng-2]<<8)+thebuf[leng-1])) //check the serial data
    {
        receiveSum = 0;
        receiveflag = 1;
    }
    return receiveflag;
}

int transmitPM01(unsigned char *thebuf)
{
    int PM01Val;
    PM01Val=((thebuf[3]<<8) + thebuf[4]); //count PM1.0 value of the air detector module
```

Configuration code and data transformation for particulate matter sensor

```

return PM01Val;
}

//transmit PM Value to PC
int transmitPM2_5(unsigned char *thebuf)
{
    int PM2_5Val;
    PM2_5Val=((thebuf[5]<<8) + thebuf[6]); //count PM2.5 value of the air detector module
    return PM2_5Val;
}

//transmit PM Value to PC
int transmitPM10(unsigned char *thebuf)
{
    int PM10Val;
    PM10Val=((thebuf[7]<<8) + thebuf[8]); //count PM10 value of the air detector module
    return PM10Val;
}

// Finish PM2.5/5/10 sensor finish

```

This code enables the DIY v2 sensor to send data to the IoT platform.

4.6.3 Cayenne IoT platform

The Cayenne platform comes with quick integration elements, which allows you to add values, graphs, or tables to visualise the data sent by the devices.

To add values or tables using the data from our sensor, you must first configure how this data will be sent in the code.

```

Cayenne. virtualWrite(0, temp, TYPE_TEMPERATURE, UNIT_CELSIUS);
    Cayenne. virtualWrite(1, press/100.0, TYPE_BAROMETRIC_PRESSURE,
UNIT_HECTOPASCAL);
    Cayenne.virtualWrite(2High);
    Cayenne. virtualWrite(3, humi, "Humidity", "%");
    Cayenne. virtualWrite(4, PM01Value);
    Cayenne. virtualWrite(5, PM2_5Value, "Particular Material", "ug/m3");
    Cayenne. virtualWrite(6, PM10Value);
    Cayenne. virtualWrite(7, tvoc_ppb, "Volatile Organic", "Parts per Billion");
    Cayenne. virtualWrite(8, co2_eq_ppm, "Carbon Dioxide", "Parts per Million");

```

Method for sending data to Cayenne. (element,value)

The configuration consists of calling the Cayenne.virtualWrite(channel,value) function. With this, the platform can call the items and visualise their value. There are pre-created elements within the platform, such as the following:

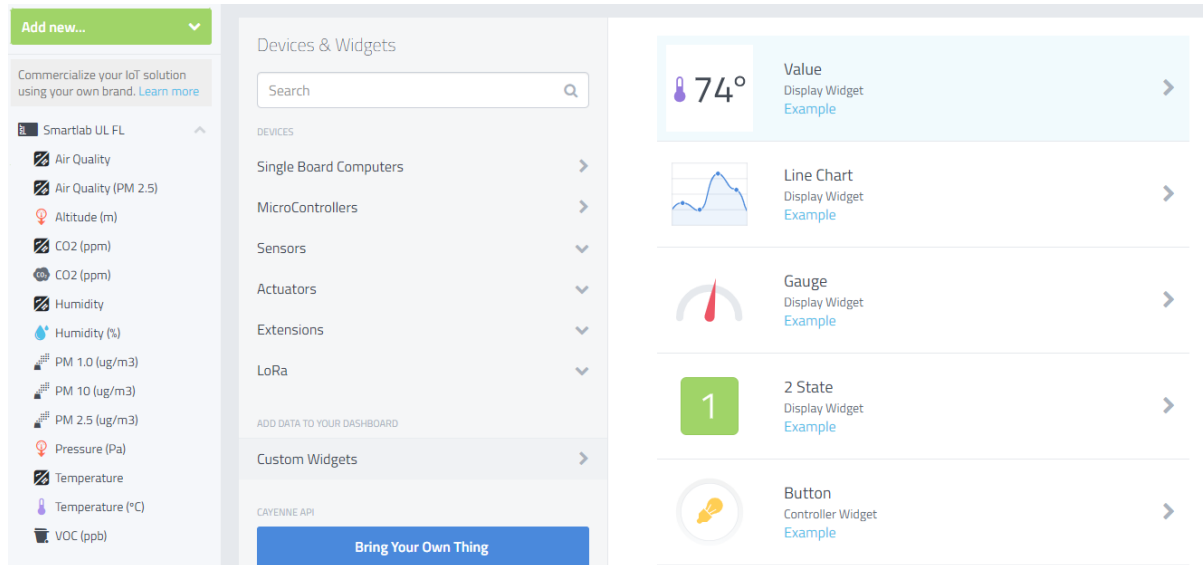



Fig. 51: Cayenne IoT Dashboard Widget creation part I

Elements such as:

- Value
- Charts
- Gauge
- State
- Button


The settings for each item are similar:

Enter Settings

 74°

Value
Display Widget

Device

 Smartlab UL FL

Sensor

Data

Unit

Channel

Choose Icon

Step 1: Code

Add Widget

[Docs: Using MQTT with Cayenne](#)

Fig. 52: Cayenne IoT Dashboard Widget creation part II

The important thing is the "channel" box. That is where we should write the channel through which the data from our sensor comes. For example:

```
Cayenne. virtualWrite(5, PM2_5Value, "Particular Material", "ug/m3");  
Cayenne. virtualWrite(6, PM10Value);
```

Through channel 5 comes the data from the particulate matter sensor, specifically the value 2.5. On the other hand, in channel 6 comes the data from the same sensor, specifically those of value 10.

This process is repeated for each element you want to place in the dashboard. This is an example of all the data arranged in values, gauges and charts.

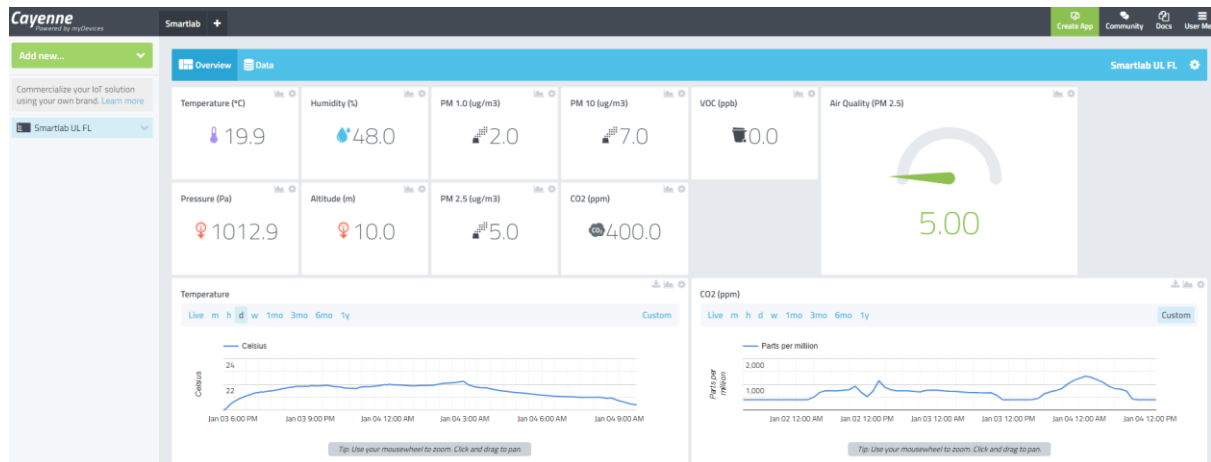


Fig. 52: Cayenne IoT Dashboard Widget creation part III

Each element we add on the screen has a record in the platform's database and can be accessed from the data icon at the top, or in each component, select the data icon. Select the time of the sample and begin the download of an Excel document.

Autoguardado Data: 1672844372703								
Buscar								
Archivo Inicio Insertar Disposición de página Fórmulas Datos Revisar Vista Automatizar Complementos Ayuda								
Deshacer Portapapeles Fuente Alineación Número Estilos								
Timestamp								
A	B	C	D	E	F	G	H	
Timestamp	DeviceID	Channel	SensorName	SensorID	DataType	Unit	Value	
2023-01-03T15:00:00.000Z	3d9f2170-8063-11ed-b193-d9789b2af62b	8 CO2 (ppm)	27854dd0-8075-11ed-8d53-d7cd1025126a	co2	Parts per Mil	400		
2023-01-03T15:01:00.000Z	3d9f2170-8063-11ed-b193-d9789b2af62b	8 CO2 (ppm)	27854dd0-8075-11ed-8d53-d7cd1025126a	co2	Parts per Mil	400		
2023-01-03T15:02:00.000Z	3d9f2170-8063-11ed-b193-d9789b2af62b	8 CO2 (ppm)	27854dd0-8075-11ed-8d53-d7cd1025126a	co2	Parts per Mil	400		
2023-01-03T15:03:00.000Z	3d9f2170-8063-11ed-b193-d9789b2af62b	8 CO2 (ppm)	27854dd0-8075-11ed-8d53-d7cd1025126a	co2	Parts per Mil	400		
2023-01-03T15:04:00.000Z	3d9f2170-8063-11ed-b193-d9789b2af62b	8 CO2 (ppm)	27854dd0-8075-11ed-8d53-d7cd1025126a	co2	Parts per Mil	400		
2023-01-03T15:05:00.000Z	3d9f2170-8063-11ed-b193-d9789b2af62b	8 CO2 (ppm)	27854dd0-8075-11ed-8d53-d7cd1025126a	co2	Parts per Mil	400		
2023-01-03T15:06:00.000Z	3d9f2170-8063-11ed-b193-d9789b2af62b	8 CO2 (ppm)	27854dd0-8075-11ed-8d53-d7cd1025126a	co2	Parts per Mil	400		
2023-01-03T15:07:00.000Z	3d9f2170-8063-11ed-b193-d9789b2af62b	8 CO2 (ppm)	27854dd0-8075-11ed-8d53-d7cd1025126a	co2	Parts per Mil	400		
2023-01-03T15:08:00.000Z	3d9f2170-8063-11ed-b193-d9789b2af62b	8 CO2 (ppm)	27854dd0-8075-11ed-8d53-d7cd1025126a	co2	Parts per Mil	400		
2023-01-03T15:09:00.000Z	3d9f2170-8063-11ed-b193-d9789b2af62b	8 CO2 (ppm)	27854dd0-8075-11ed-8d53-d7cd1025126a	co2	Parts per Mil	400		
2023-01-03T15:10:00.000Z	3d9f2170-8063-11ed-b193-d9789b2af62b	8 CO2 (ppm)	27854dd0-8075-11ed-8d53-d7cd1025126a	co2	Parts per Mil	400		
2023-01-03T15:11:00.000Z	3d9f2170-8063-11ed-b193-d9789b2af62b	8 CO2 (ppm)	27854dd0-8075-11ed-8d53-d7cd1025126a	co2	Parts per Mil	400		
2023-01-03T15:12:00.000Z	3d9f2170-8063-11ed-b193-d9789b2af62b	8 CO2 (ppm)	27854dd0-8075-11ed-8d53-d7cd1025126a	co2	Parts per Mil	400		
2023-01-03T15:13:00.000Z	3d9f2170-8063-11ed-b193-d9789b2af62b	8 CO2 (ppm)	27854dd0-8075-11ed-8d53-d7cd1025126a	co2	Parts per Mil	400		
2023-01-03T15:14:00.000Z	3d9f2170-8063-11ed-b193-d9789b2af62b	8 CO2 (ppm)	27854dd0-8075-11ed-8d53-d7cd1025126a	co2	Parts per Mil	400		
2023-01-03T15:15:00.000Z	3d9f2170-8063-11ed-b193-d9789b2af62b	8 CO2 (ppm)	27854dd0-8075-11ed-8d53-d7cd1025126a	co2	Parts per Mil	400		
2023-01-03T15:16:00.000Z	3d9f2170-8063-11ed-b193-d9789b2af62b	8 CO2 (ppm)	27854dd0-8075-11ed-8d53-d7cd1025126a	co2	Parts per Mil	400		
2023-01-03T15:17:00.000Z	3d9f2170-8063-11ed-b193-d9789b2af62b	8 CO2 (ppm)	27854dd0-8075-11ed-8d53-d7cd1025126a	co2	Parts per Mil	400		
2023-01-03T15:18:00.000Z	3d9f2170-8063-11ed-b193-d9789b2af62b	8 CO2 (ppm)	27854dd0-8075-11ed-8d53-d7cd1025126a	co2	Parts per Mil	400		
2023-01-03T15:19:00.000Z	3d9f2170-8063-11ed-b193-d9789b2af62b	8 CO2 (ppm)	27854dd0-8075-11ed-8d53-d7cd1025126a	co2	Parts per Mil	400		
2023-01-03T15:20:00.000Z	3d9f2170-8063-11ed-b193-d9789b2af62b	8 CO2 (ppm)	27854dd0-8075-11ed-8d53-d7cd1025126a	co2	Parts per Mil	400		
2023-01-03T15:21:00.000Z	3d9f2170-8063-11ed-b193-d9789b2af62b	8 CO2 (ppm)	27854dd0-8075-11ed-8d53-d7cd1025126a	co2	Parts per Mil	400		
2023-01-03T15:22:00.000Z	3d9f2170-8063-11ed-b193-d9789b2af62b	8 CO2 (ppm)	27854dd0-8075-11ed-8d53-d7cd1025126a	co2	Parts per Mil	400		
2023-01-03T15:23:00.000Z	3d9f2170-8063-11ed-b193-d9789b2af62b	8 CO2 (ppm)	27854dd0-8075-11ed-8d53-d7cd1025126a	co2	Parts per Mil	400		
2023-01-03T15:24:00.000Z	3d9f2170-8063-11ed-b193-d9789b2af62b	8 CO2 (ppm)	27854dd0-8075-11ed-8d53-d7cd1025126a	co2	Parts per Mil	400		
2023-01-03T15:25:00.000Z	3d9f2170-8063-11ed-b193-d9789b2af62b	8 CO2 (ppm)	27854dd0-8075-11ed-8d53-d7cd1025126a	co2	Parts per Mil	400		
2023-01-03T15:26:00.000Z	3d9f2170-8063-11ed-b193-d9789b2af62b	8 CO2 (ppm)	27854dd0-8075-11ed-8d53-d7cd1025126a	co2	Parts per Mil	400		
2023-01-03T15:27:00.000Z	3d9f2170-8063-11ed-b193-d9789b2af62b	8 CO2 (ppm)	27854dd0-8075-11ed-8d53-d7cd1025126a	co2	Parts per Mil	400		
2023-01-03T15:28:00.000Z	3d9f2170-8063-11ed-b193-d9789b2af62b	8 CO2 (ppm)	27854dd0-8075-11ed-8d53-d7cd1025126a	co2	Parts per Mil	400		
2023-01-03T15:29:00.000Z	3d9f2170-8063-11ed-b193-d9789b2af62b	8 CO2 (ppm)	27854dd0-8075-11ed-8d53-d7cd1025126a	co2	Parts per Mil	400		
2023-01-03T15:30:00.000Z	3d9f2170-8063-11ed-b193-d9789b2af62b	8 CO2 (ppm)	27854dd0-8075-11ed-8d53-d7cd1025126a	co2	Parts per Mil	400		
2023-01-03T15:31:00.000Z	3d9f2170-8063-11ed-b193-d9789b2af62b	8 CO2 (ppm)	27854dd0-8075-11ed-8d53-d7cd1025126a	co2	Parts per Mil	400		
2023-01-03T15:32:00.000Z	3d9f2170-8063-11ed-b193-d9789b2af62b	8 CO2 (ppm)	27854dd0-8075-11ed-8d53-d7cd1025126a	co2	Parts per Mil	400		

References

- Crabtree, T. S., McLay, A., & Wilmot, E. G. (2019). DIY artificial pancreas systems: here to stay? *Practical Diabetes*, 36(2), 63–68. <https://doi.org/10.1002/pdi.2216>
- Finn, D. (2014). DIY urbanism: implications for cities. *Journal of Urbanism: International Research on Placemaking and Urban Sustainability*, 7(4), 381–398. <https://doi.org/10.1080/17549175.2014.891149>
- Lindtner, S. (2015). Hacking with Chinese Characteristics. *Science, Technology, & Human Values*, 40(5), 854–879. <https://doi.org/10.1177/0162243915590861>
- O'Flaherty, C. (2023). W2-D2: Approved List of Sensor Devices to Match the Data Requirements under the SRI and Enable Smart Building Monitoring across the Living Lab including through DIY Kits. University of Limerick. Report. <https://doi.org/10.34961/researchrepository-ul.22770353.v1>
- Trejo Rangel, M. A., Lyes, M., Nuñez, D., Fitzgerald, H., & Kinsella, S. (2022). WP1-D1 Plan for Stakeholder Engagement and SMARTLAB Journeys Including SMARTLAB Calendar of Events. University of Limerick. <https://doi.org/https://doi.org/10.34961/researchrepository-ul.22202677.v1>
- Trejo-Rangel, M. A., Lyes, M., Fitzgerald, H., & Kinsella, S. (2023). WP1-D3: Barriers to Improving the Smartness of Buildings and Associated Barriers to the Deployment of Smart Technologies and Services. University of Limerick. <https://doi.org/https://doi.org/10.34961/researchrepository-ul.22561156.v1>
- Wolf, M., & McQuitty, S. (2011). Understanding the do-it-yourself consumer: DIY motivations and outcomes. *AMS Review*, 1(3–4), 154–170. <https://doi.org/10.1007/s13162-011-0021-2>