Sistemas industriales

Universitat d'Alacant
Universidad de Alicante

Parte 1

Francisco Joaquín Murcia Gómez

25 de febrero de 2022

Índice

1. Descripción de la practica

El objetivo de la practica seria familiarizarse con los protocolos IoT existentes, así como la obtención de datos de internet o de sensores físicos para su posterior procesamiento o muestreo mediante paneles gráficos.

Para ello se usara una Raspberry pi como base para obtener dichos datos y Node-Red par realizar las implementaciones.

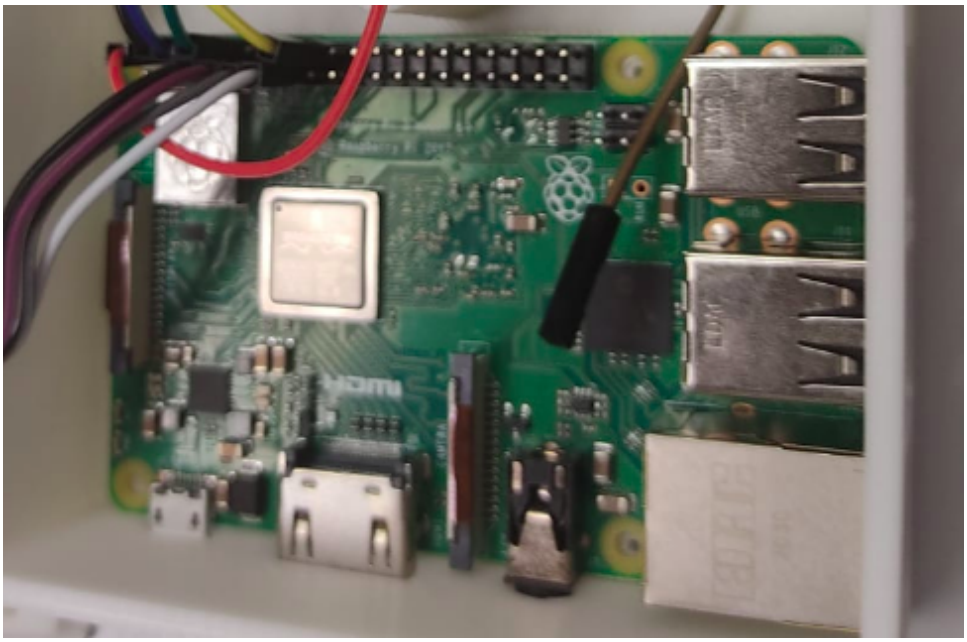

Figura 1: Raspberry pi usada

2. Lectura de datos de Open Weather

Nos hemos conectado a dos APIs libres del proveedor Open Weather(`https://openweathermap.org/`) para poder obtener datos meteorológicos, una para saber la temperatura y otra para saber la cantidad de monóxido de carbono, ambas en la ciudad de Alicante.

2.1. API de temperatura

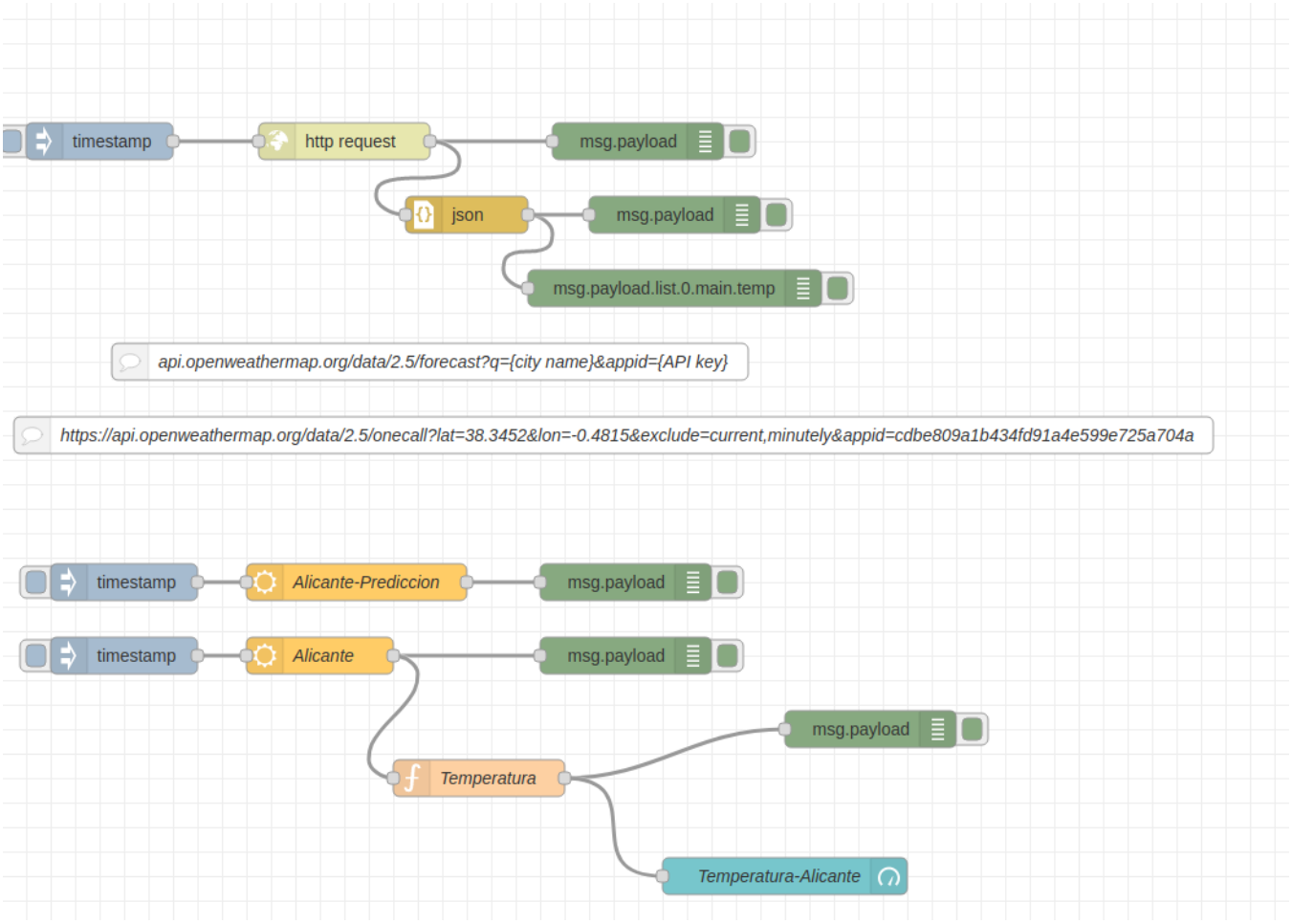

Figura 2: Nodos del sistema

Con Node-Red nos conectamos a Open Weather utilizando un token y extraemos un array que contiene la temperatura en Alicante, con una función extraemos la temperatura y lo mostramos en una dashboard.

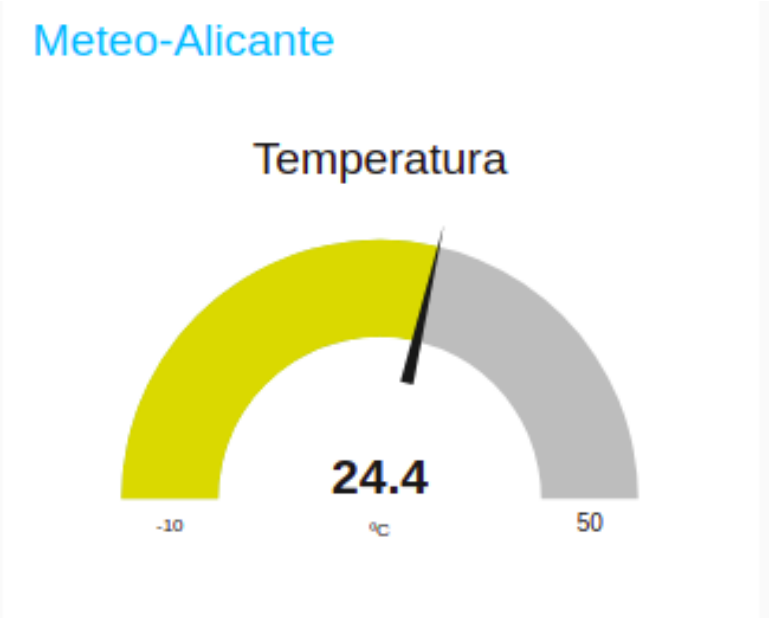

Figura 3: Dashboard de la temperatura en Alicante

2.2. API de calidad del aire (Monóxido de carbono)

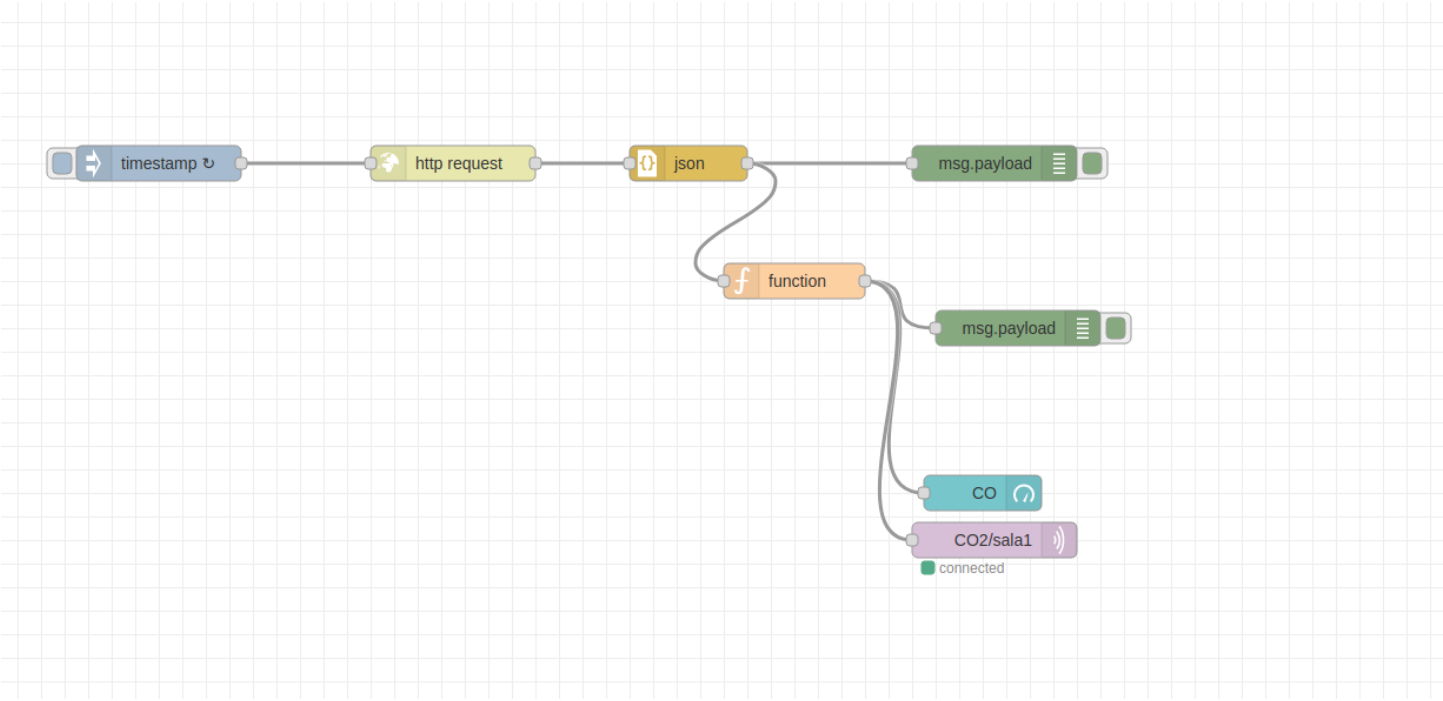

Figura 4: Implementación en Node-Red

Nos hemos conectado a otra API de Open Weather donde extraemos el nivel de monóxido de carbono en Alicante. Es muy similar a la anterior pero en este enviamos los datos mediante el protocolo MQTT para que la clase pueda leerlos, para ello nos conectamos al Broker de la clase y publicamos allí los datos.

3. Lectura de datos de ESIOS

Nos hemos conectado al servicio de información de la red eléctrica Española "ESIOS" (`https://www.esios.ree.es/es/pvpc`) para poder obtener datos del precio de la luz máximo y el mínimo actual.

Para ello utilizando un token que obtenemos solicitándoselo

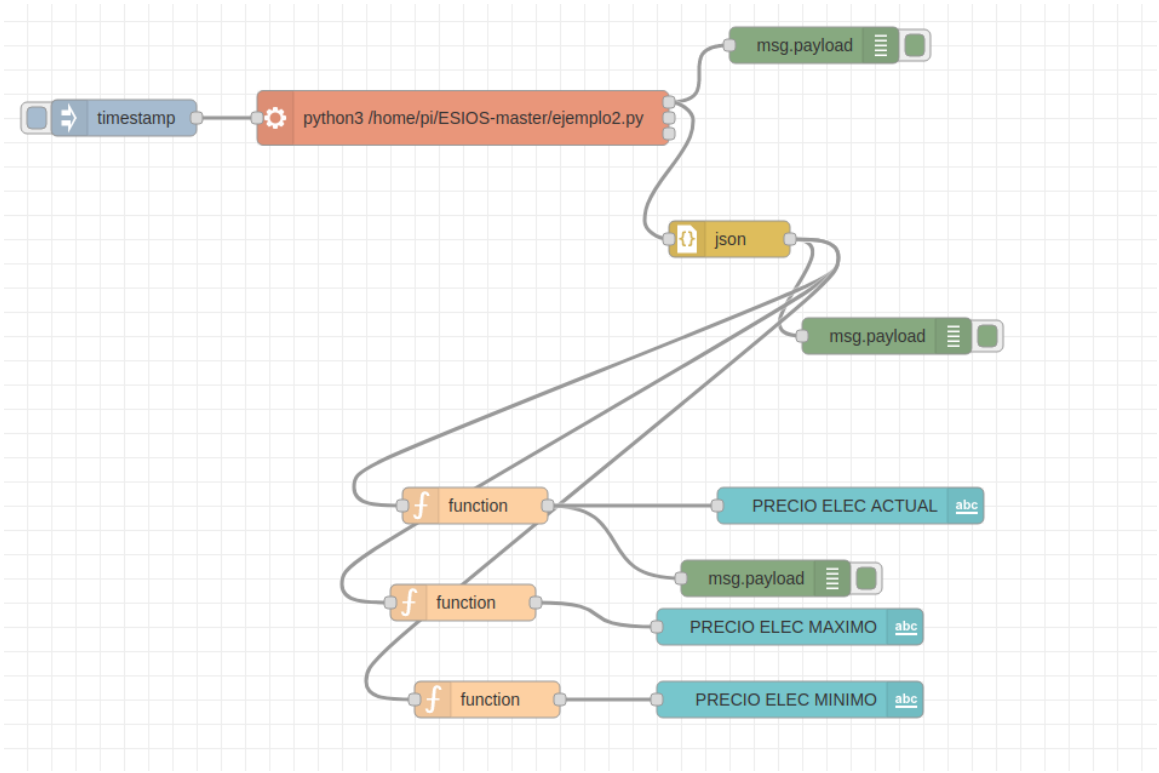

Figura 5: Implementación en Node-Red

Hemos necesitado un código en python para poder conectarnos a la api, este nos devuelve un JSON con los precios, este JSON los pasamos por tres funciones, obteniendo así el precio actual, el máximo y el mínimo y finalmente se envía a su dashboard correspondiente, viéndose de la siguiente manera:

Figura 6: Dashboard precio MW

4. Lectura de datos del sensor BLE

Nos hemos conectado a un sensor BLE por bluetooth para extraer datos como en este caso temperatura y humedad, para ello hemos usado el siguiente esquema de Node-Red y codigo en python:

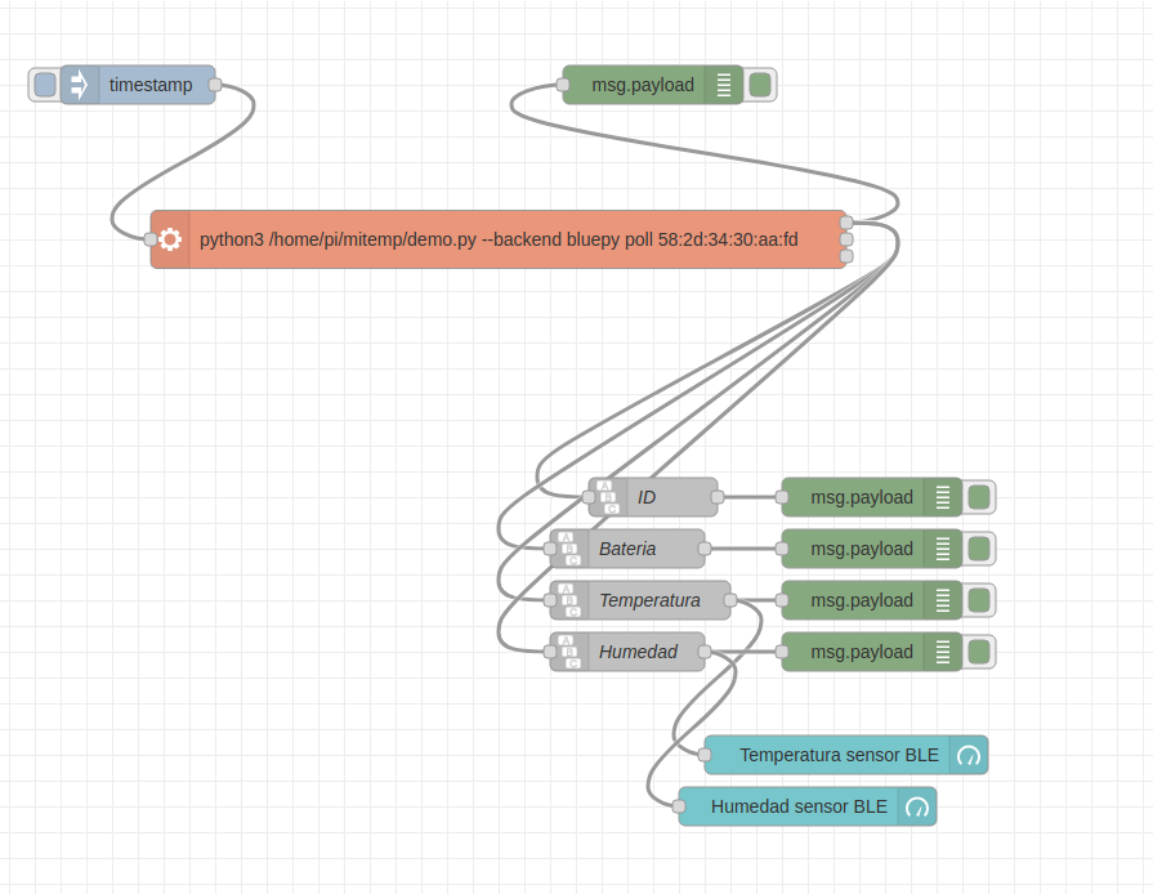

Figura 7: Implementación en Node-Red

```
usr/bin/env python3
"""Demo file showing how to use the mitemp library."""

import argparse
import re
import logging
import sys

from btlewrap import available_backends, BluepyBackend, GatttoolBackend, PygattBackend
from mitemp_bt.mitemp_bt_poller import MiTempBtPoller, \
    MI_TEMPERATURE, MI_HUMIDITY, MI_BATTERY


def valid_mitemp_mac(mac, pat=re.compile(r"[0-9A-F]{2}:[0-9A-F]{2}:[0-9A-F]{2}:[0-9A-F]{2}:[0-9A-F]{2}:[0-9A-F]{2}")):
    """Check for valid mac adresses."""
    if not pat.match(mac.upper()):
        raise argparse.ArgumentTypeError('The MAC address "{}" seems to be in the wrong format'.format(mac))
    return mac
```

```
19
20
21  def poll(args):
22      """Poll data from the sensor."""
23      backend = _get_backend(args)
24      poller = MiTempBtPoller(args.mac, backend)
25      print("Getting data from Mi Temperature and Humidity Sensor")
26      print("FW: {}".format(poller.firmware_version()))
27      print("Name: {}".format(poller.name()))
28      print("Battery: {}".format(poller.parameter_value(MI_BATTERY)))
29      print("Temperature: {}".format(poller.parameter_value(MI_TEMPERATURE)))
30      print("Humidity: {}".format(poller.parameter_value(MI_HUMIDITY)))
31
32
33  # def scan(args):
34  #     """Scan for sensors."""
35  #     backend = _get_backend(args)
36  #     print('Scanning for 10 seconds...')
37  #     devices = mitemp_scanner.scan(backend, 10)
38  #     devices = []
39  #     print('Found {} devices:'.format(len(devices)))
40  #     for device in devices:
41  #         print('  {}'.format(device))
42
43
44  def _get_backend(args):
45      """Extract the backend class from the command line arguments."""
46      if args.backend == 'gatttool':
47          backend = GatttoolBackend
48      elif args.backend == 'bluepy':
49          backend = BluepyBackend
50      elif args.backend == 'pygatt':
51          backend = PygattBackend
52      else:
53          raise Exception('unknown backend: {}'.format(args.backend))
54      return backend
55
56
57  def list_backends(_):
58      """List all available backends."""
59      backends = [b.__name__ for b in available_backends()]
60      print('\n'.join(backends))
61
62
63  def main():
64      """Main function.
65
66      Mostly parsing the command line arguments.
67      """
68      parser = argparse.ArgumentParser()
69      parser.add_argument('--backend', choices=['gatttool', 'bluepy', 'pygatt'], default='gatttool')
70      parser.add_argument('-v', '--verbose', action='store_const', const=True)
71      subparsers = parser.add_subparsers(help='sub-command help', )
72
73      parser_poll = subparsers.add_parser('poll', help='poll data from a sensor')
74      parser_poll.add_argument('mac', type=valid_mitemp_mac)
75      parser_poll.set_defaults(func=poll)
```

```
76
77      # parser_scan = subparsers.add_parser('scan', help='scan for devices')
78      # parser_scan.set_defaults(func=scan)
79
80      parser_scan = subparsers.add_parser('backends', help='list the available backends')
81      parser_scan.set_defaults(func=list_backends)
82
83      args = parser.parse_args()
84
85      if args.verbose:
86          logging.basicConfig(level=logging.DEBUG)
87
88      if not hasattr(args, "func"):
89          parser.print_help()
90          sys.exit(0)
91
92      args.func(args)
93
94
95  if __name__ == '__main__':
96      main()
```

Dando como resultado la siguiente dasboard:

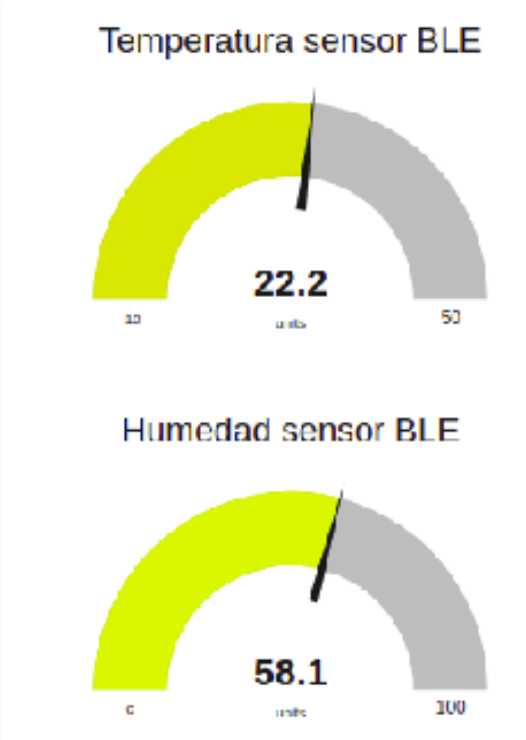

Figura 8: Dasboard de la temperatura y humedad

5. Medidor de dioxido de carbono

5.1. Descripción

El sistema monitorea el nivel de CO2 cada 20 segundos y dependiendo de las partes por millón detectadas se enciende una luz de verde (menos de 400 ppm), de azul (400-800 ppm) o de rojo (mas de 800 ppm).

Atendiendo a la implementación hardware, se ha conectado un sensor de CO2 y un led RGB a la Raspberry pi.

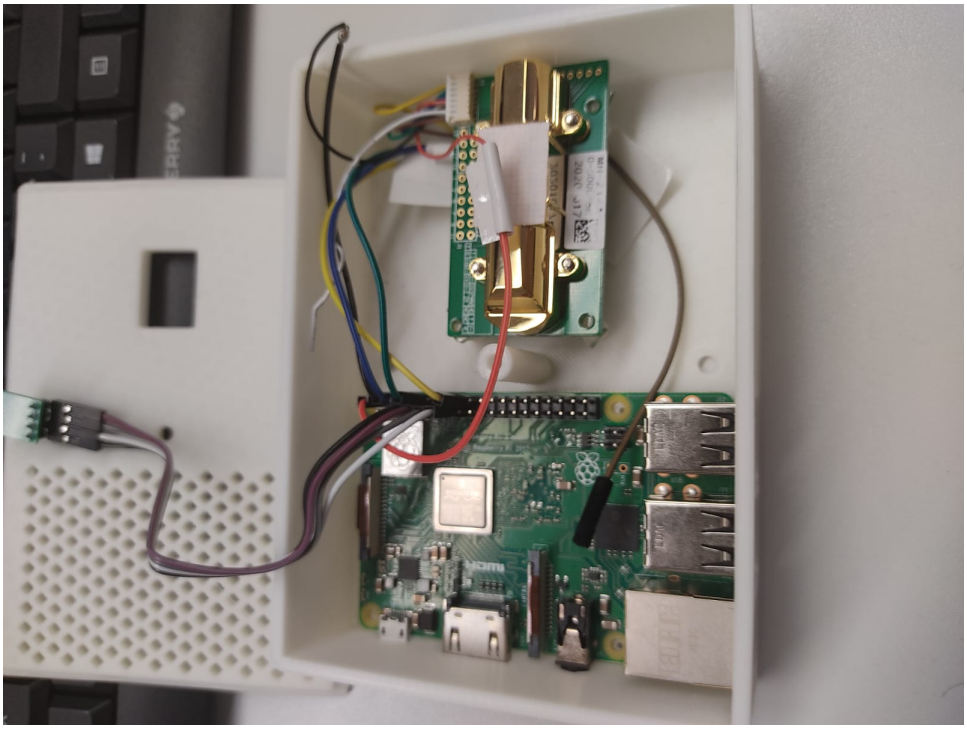

Figura 9: Raspberry pi (abajo), sensor CO2 (arriba) y led RGB (izquierda)

5.2. Implementación

Para la lectura de los datos, hemos usado el siguiente código en Python que obtiene las partes por millón y la hora en que fue la medición. Estos los devuelven en formato JSON y enciende el led en el valor correspondiente.

```
1  import serial
2  import time
3  import json
4  import RPi.GPIO as GPIO
5  import time
6
7  # Define los colores en modo hexadecimal
8  COLORS = [0xFF0000, 0x00FF00, 0x0000FF, 0x00FFFF, 0xFF00FF, 0xFFFF00, 0xFFFFFF,
9            0xB695C0, 0xFFFF00]
10 # Establece los pines conforme a GPIO
11 pins = {'Red': 22, 'Green': 17, 'Blue': 27}
12
```

```
13
14  def mapea(x, in_min, in_max, out_min, out_max):
15      return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min
16
17
18  def set_color(color):
19      # calcula el valor para cada canal R, G, B
20      R_val = (color & 0xFF0000) >> 16
21      G_val = (color & 0x00FF00) >> 8
22      B_val = (color & 0x0000FF) >> 0
23
24      # Convierte el color de 0~255 a 0 ó 1 (entero)
25      R_val = int(mapea(R_val, 0, 255, 0, 1))
26      G_val = int(mapea(G_val, 0, 255, 0, 1))
27      B_val = int(mapea(B_val, 0, 255, 0, 1))
28
29      # asigna a cada pin el valor calculado
30      GPIO.output(pins['Red'], R_val)
31      GPIO.output(pins['Green'], G_val)
32      GPIO.output(pins['Blue'], B_val)
33
34      # imprime los resultados
35      #print("R_val = %s, G_val = %s, B_val = %s" % (R_val, G_val, B_val))
36
37
38  # quita la tensión en cada uno de los pines
39  def reset():
40      for i in pins:
41          GPIO.output(pins[i], 0)
42
43  class CO2Sensor():
44    request = [0xff, 0x01, 0x86, 0x00, 0x00, 0x00, 0x00, 0x00, 0x79]
45
46    def __init__(self, port='/dev/ttyS0'):
47      self.serial = serial.Serial(
48          port = port,
49          timeout = 1
50      )
51
52    def get(self):
53      self.serial.write(bytearray(self.request))
54      response = self.serial.read(9)
55      if len(response) == 9:
56        current_time = time.strftime('%H:%M:%S', time.localtime())
57        s = {"time": current_time, "ppa": (response[2] << 8) | response[3], "temp": response[4]}
58        d = json.dumps(s)
59        return d
60      return -1
61
62  def main():
63    # other Pi versions might need CO2Sensor('/dev/ttyAMA0')
64    sensor = CO2Sensor()
65
66      # Establece el modo, en este caso a los valores GPIO
67    GPIO.setmode(GPIO.BCM)
68    for i in pins:
69          # configura los pines como de salida
```

```
70      GPIO.setup(pins[i], GPIO.OUT, initial=GPIO.HIGH)
71    reset()
72
73    s = sensor.get()
74    decoded = json.loads(s)
75    ppa = decoded['ppa']
76    #print(ppa)
77    if ppa >= 800:
78        set_color(0x00FF00) #rojo
79    elif ppa < 400:
80        set_color(0x0000FF) #verde
81    else:
82        set_color(0x00FFFF) #verde amarillo
83    print(sensor.get())
84    time.sleep(2)
85
86  if __name__ == '__main__':
87    main()WhatsApp Image 2022-01-16 at 16.39.47.jpeg
```

Para que el código anterior utilice el led este tiene que estar conectado según el siguiente esquema:

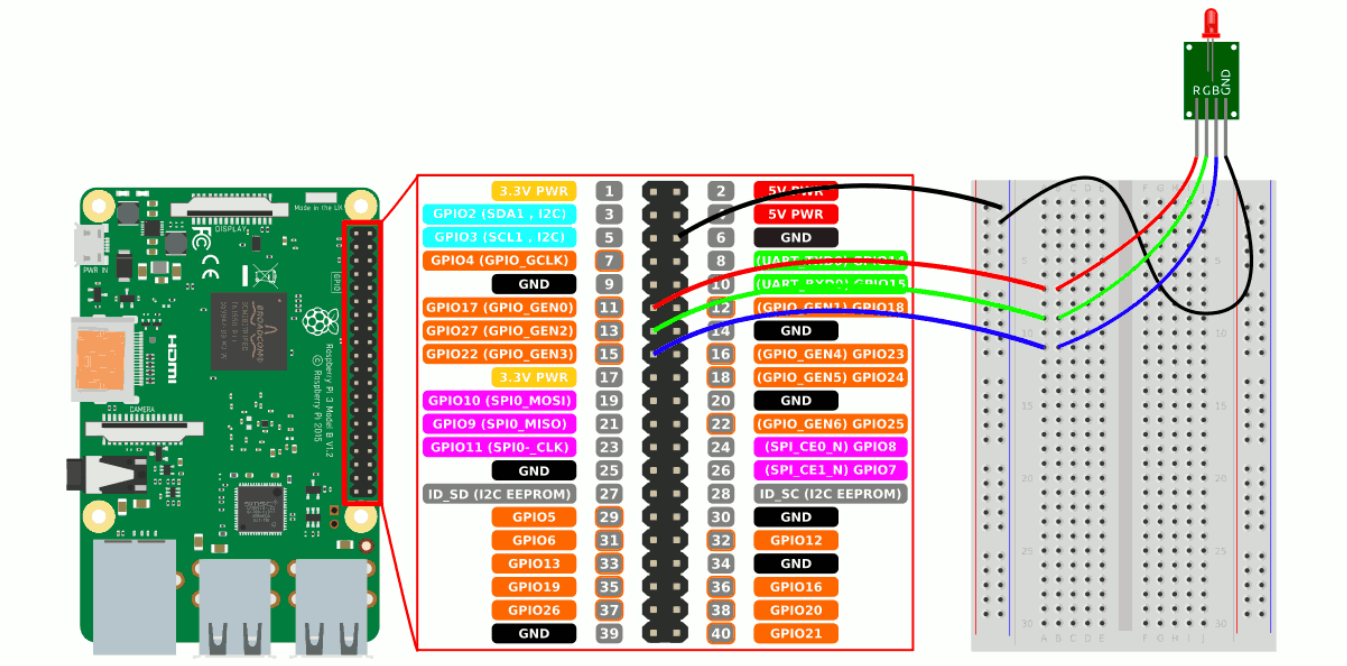

Figura 10: Esquema de pines para el led RGB

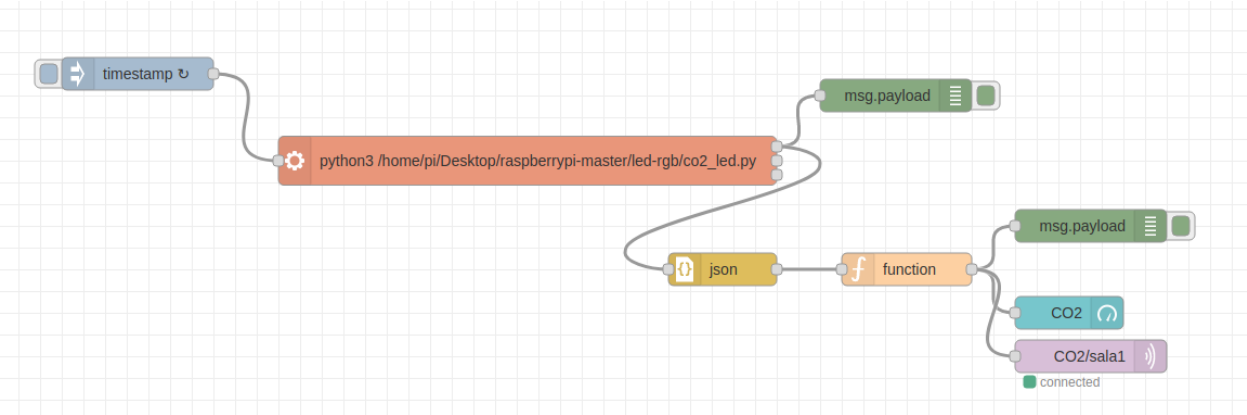

Figura 11: Nodos del sistema

Los datos obtenidos se envían mediante el protocolo MQTT al broker del aula.

5.3. Resultados

El sistema imprime por pantalla la siguiente información:

- Time: La hora de la medicion.
- PPA: Partes por millon de CO2 en el aire.

Figura 12: Resultados de la ejecución durante 36 segundos

Para ver el funcionamiento del led hemos inyectado valores de prueba (250 ppa, 450 ppa y 906 ppa), viéndose de la siguiente manera

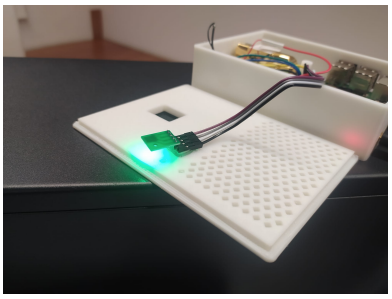

Figura 13: 250 ppa, led verde

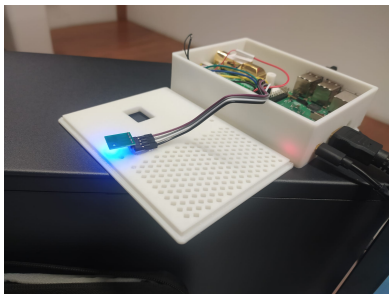

Figura 14: 450 ppa, led azul

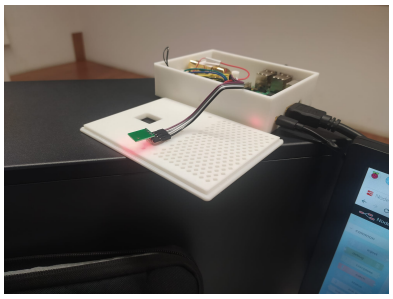

Figura 15: 906 ppa, led rojo

A parte de gestionar un LED la información se muestra en una dashboard.

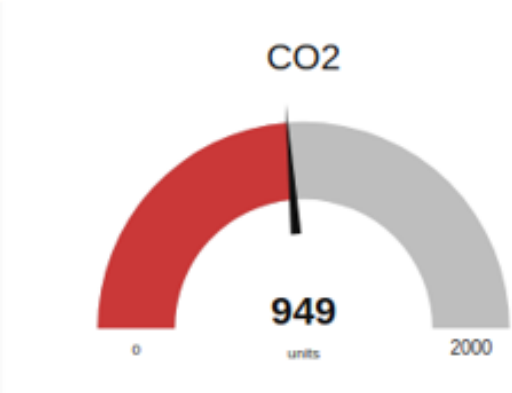

Figura 16: Dashboard de CO2 con ppa 949

6. Obtención de datos mediante el protocolo MQTT

Como ultima implementación obtenemos datos que otros compañeros compartían mediante MQTT, para ello nos hemos conectado al broker y nos suscribimos a la temperatura y la humedad que envían nuestros compañeros .

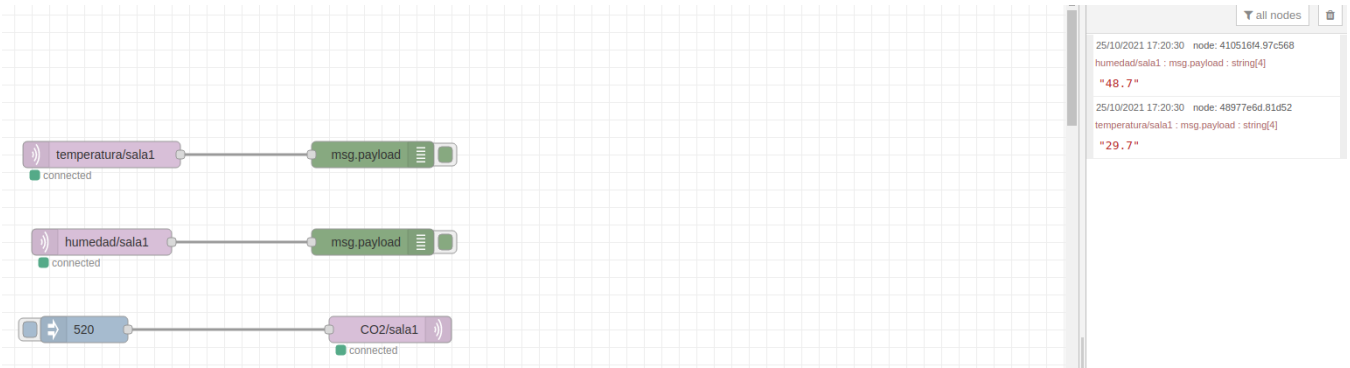

Figura 17: Implementación en Node-Red (izquierda) y datos obtenidos (derecha)

7. Ampliación de la asignatura

Para la ampliación de la asignatura se ha creado un BOT para Telegram llamado "MeteoAlicanteBot" donde podremos consultar tanto la información meteorológica como la polución en el aire. Esta información también la subimos a Ubidots, una plataforma IoT para la visualización y análisis de datos.

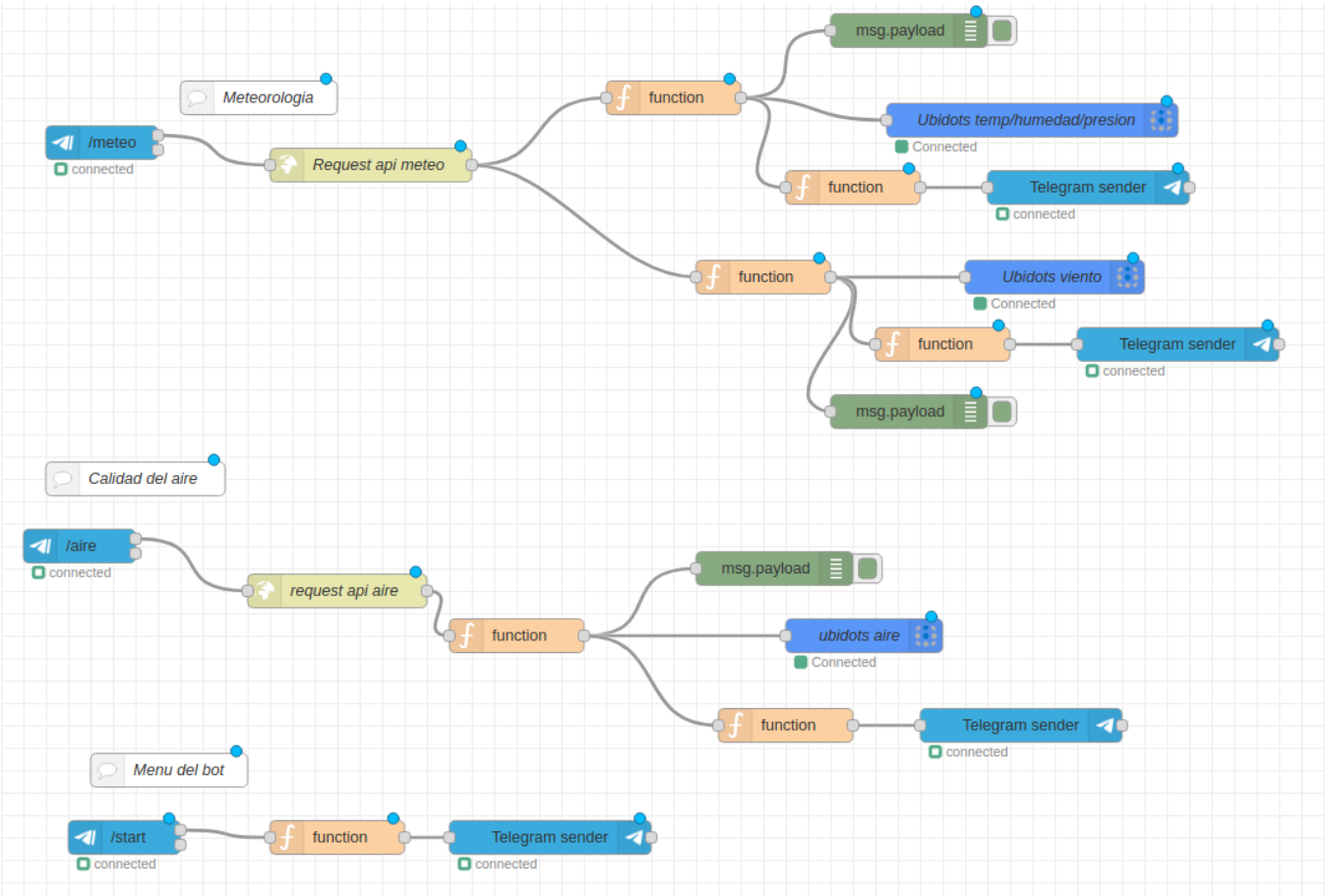

Figura 18: Implementacion en Node-red

7.1. Lectura de datos

Para la lectura de datos se ha reusado las apis creadas en el apartado 2.1 y 2.2 pero hemos ampliado los datos para obtener no solo la temperatura y el nivel de Monóxido de carbono. de tal forma que en la api 2.1 obtenemos la temperatura, sensación térmica, humedad y presión del aire y de la 2.2 los parámetros del Índice de Calidad del Aire (Monóxido de carbono, Dióxido de azufre, Dióxido de Nitrógeno, Partículas menores a 10 micrómetros, Partículas menores a 2,5 micrómetros, Ozono troposférico), Monóxido de Carbono, Amoniaco, Monóxido de Nitrógeno y Dióxido de Azufre.

7.2. Ubidots

Para el analisis de los datos se ha empleado la plataforma IoT que se comento en clase llamada Ubidots. En la plataforma recibimos los datos y los mostramos gráficamente. Para enviar los datos hacemos uso delos bloques que provee Ubidots que no son mas que una facilitación del protocolo MQTT.

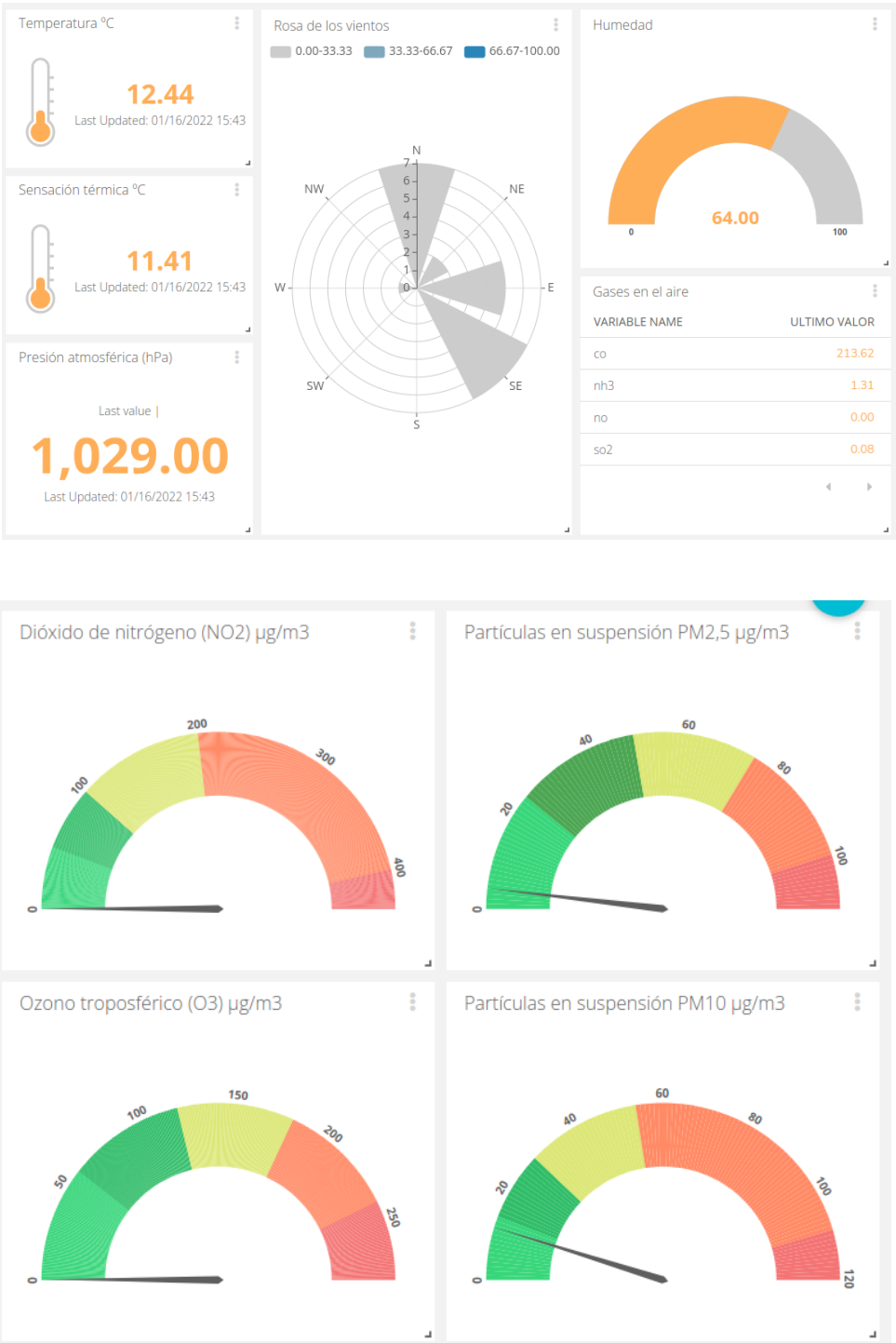

Figura 19: Dashboard creado en Ubidots

Como podemos observar tendríamos la temperatura la presión atmosférica, una rosa de los viento donde aparecen las rachas de viento en km/h, la humedad, los gases contaminantes y finalmente los parámetros del ICA (Índice de Calidad del Aire)

7.3. Bot de Telegram

Se ha creado un simple Bot donde con la instrucción "/start"nos despliega dos opciones para consultar, la calidad del aire '/aire" y la meteorologia "/meteo"

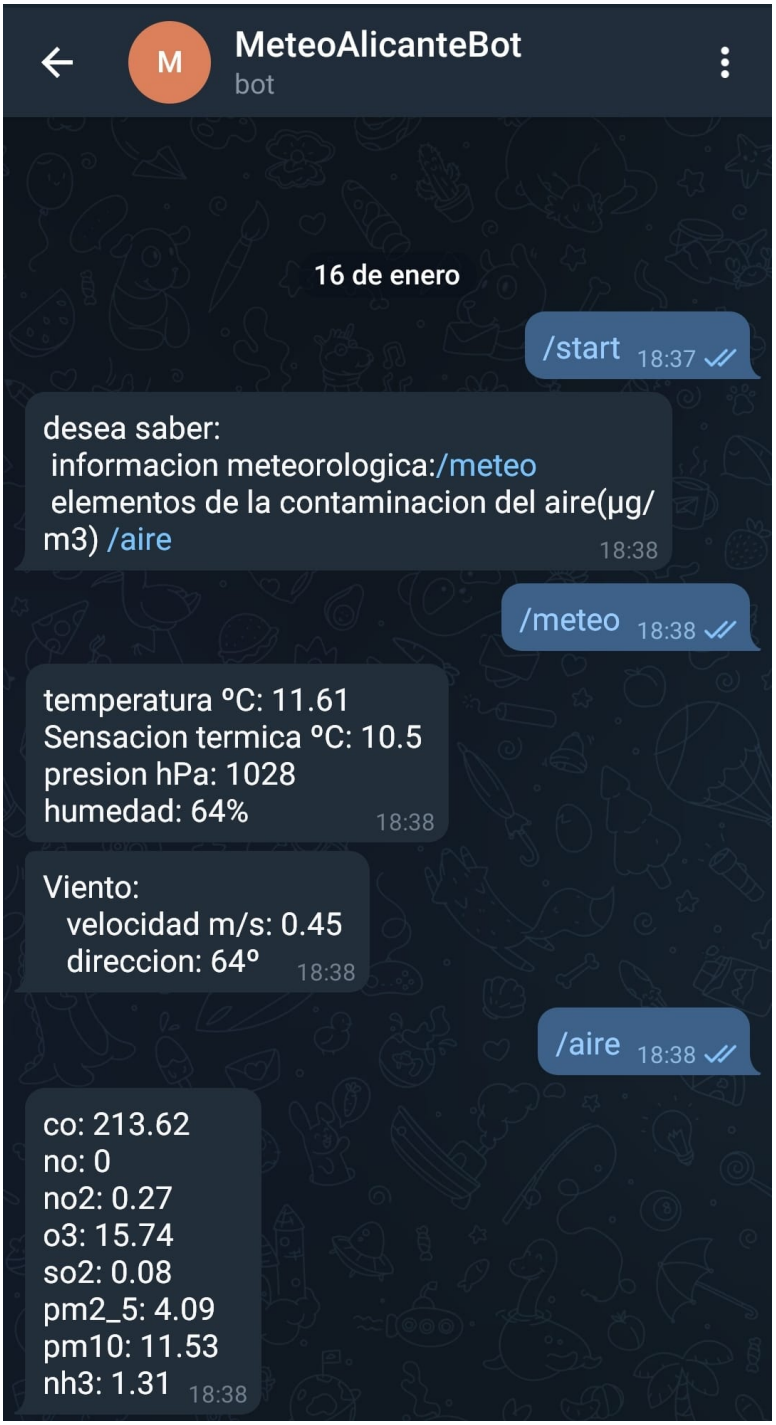

Figura 20: Funcionamiento de MeteoAlicanteBot