

Instructivo montaje HW+IDE Arduino+VSC/PJO -- actualizado 07-10-20

1. Montaje de Hardware ESP32+DHT22

Se describe el montaje usando placas NodeMCU ESP32 de DOIT-devkit-v1, 15 pines por lado.

1.1) Para el montaje se usó un ESP32 tipo DOIT-devkit-v1 (15 pines por lado) adquirido a Starware en setiembre. Hay muchos modelos distintos por lo cual esta conexión dependerá de la placa adquirida. La conexión es similar a la indicada por el PPT Clase5 / hoja 13 de la cátedra de DapIoT (Figura 1), solo que la que se indica es una placa de 18 pines por lado, y en el diagrama se utiliza GPIO4 para comunicar con el DHT. En nuestro caso se usó un DHT22 que viene marcado como AM2302. En la Figura 2 se ve el diagrama cableado, con marca del pin1 del sensor. La secuencia es Pin 1=Vcc (3V3, hasta 5V no hay problema); Pin 2 = Data; Pin 3 no conectado; Pin4 = GND. Se utilizó una resistencia de pull-up de 10K a 3V3 desde Data.

Dispositivo IoT con ESP32 y DHT11/DHT22

Conexiones eléctricas:

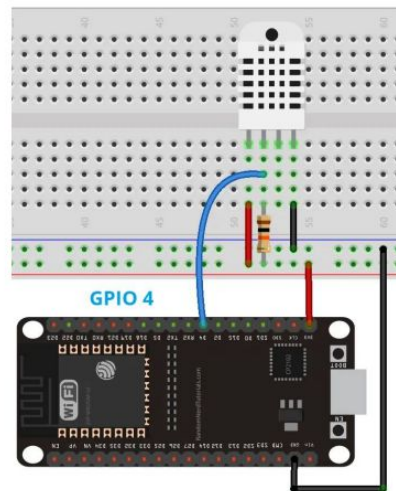


Figura 1 – Diagrama sugerido por cátedra DapIoT

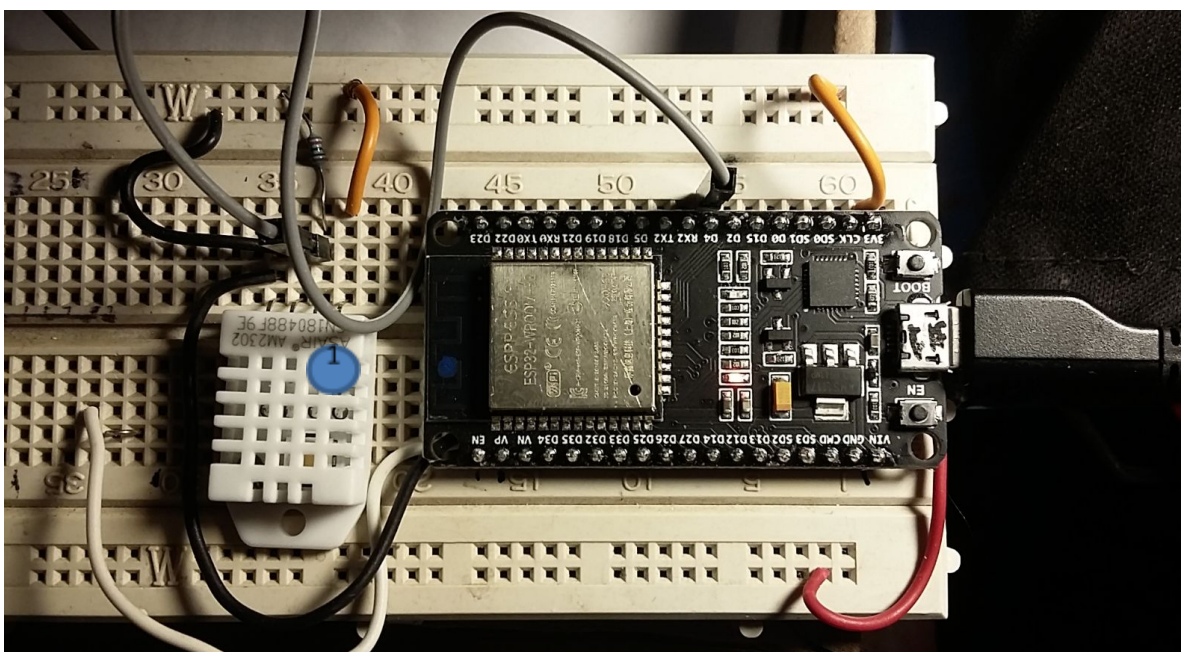


Figura 2 – Vista del montaje de placas, con marca del pin1 del DHT22

1.2) El diagrama general de conexión a la RBPi para establecer la comunicación MQTT requerida en el TP se muestra en la Figura 3. Se muestra la Base SQLite cuya conexión viene pre-cargada en la imagen SD remitida por la Cátedra, pero se puede elegir otra base de datos. Se describe la comunicación con el sistema en la RBPi, donde hay que instalar el Broker Mosquitto, y Node-red (en la imagen viene instalado). Esto permite la visualización de ambas magnitudes de Temperatura y Humedad en el dashboard de node-red. Se incluye en el repo [1] el código de: la versión Arduino con los cambios que se requirieron para que funcione, la versión VSC-PlatformIO (similar a lo que se utilizó con el docente A.Bassi en la parte MQTTConnection, en Arquitectura de Protocolos), y el flow de NodeRed actualizado para leer dos canales de MQTT-in (LED aun no implementado) y visualización de Temperatura y Humedad en dos gauges usando node-red dashboard.

Diagrama 09-20

Ingreso Datos DAIoT

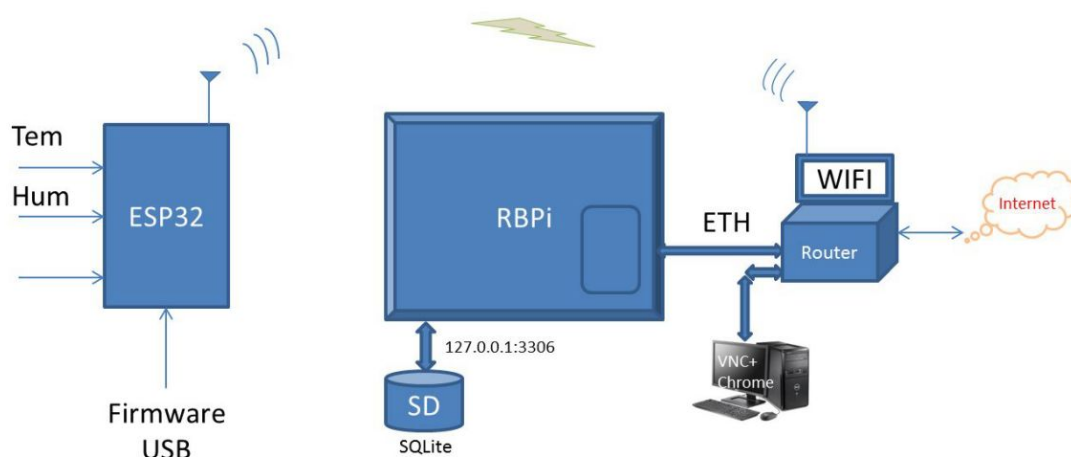


Figura 3 – Diagrama general / conexión a RBPi y visualización de datos (preliminar).

2. Programación vía Arduino IDE

2.2) Aquí se tomó como base el archivo .ino suministrado por la cátedra:

https://github.com/gramoscelli/2B2020_DdAIoT.git

```

/dev/ttyUSB0
===== Message: 39.40
Publish acknowledged. packetId: 791
Publishing on topic ESP32_98gvfde7832cvssd/dht/temperature at QoS 1, packetId: 792
===== Message: 21.20
Publishing on topic ESP32_98gvfde7832cvssd/dht/humidity at QoS 1, packetId: 793
Publish acknowledged. packetId: 792
===== Message: 39.40
Publish acknowledged. packetId: 793
Publishing on topic ESP32_98gvfde7832cvssd/dht/temperature at QoS 1, packetId: 794
===== Message: 21.20
Publishing on topic ESP32_98gvfde7832cvssd/dht/humidity at QoS 1, packetId: 795
Publish acknowledged. packetId: 794
===== Message: 39.50
Publish acknowledged. packetId: 795

Autoscroll Show timestamp
Newline 115200 baud Clear output

// Led builtin
#define LED_BUILTIN 2

// Uncomment whatever DHT sensor type you're using
// #define DHTTYPE DHT11 // DHT 11
#define DHTTYPE DHT22 // DHT 22 (AM2302), AM2321 2.10.20
// #define DHTTYPE DHT21 // DHT 21 (AM2301)

Done uploading.
Auto-detected Flash size: 4MB
Compressed 8192 bytes to 47...
Wrote 8192 bytes (47 compressed) at 0x00000000 in 0.0 seconds (effective 3061.0 kbit/s)...
Hash of data verified.
Compressed 15856 bytes to 10276...
Wrote 15856 bytes (10276 compressed) at 0x00001000 in 0.1 seconds (effective 990.2 kbit/s)...
Hash of data verified.
Compressed 707712 bytes to 393976...
Wrote 707712 bytes (393976 compressed) at 0x00010000 in 6.1 seconds (effective 925.2 kbit/s)...
Hash of data verified.
Compressed 3072 bytes to 128...
Wrote 3072 bytes (128 compressed) at 0x00000000 in 0.0 seconds (effective 1488.5 kbit/s)...
Hash of data verified.
Leaving...
Hard resetting via RTS pin...
36
  
```

Figura 4 – Sistema funcionando con el .ino modificado, respecto al suministrado por la cátedra

Las modificaciones fueron menores. La selección del PIN 4 para datos del DHT, la selección del DHT que se tenga (en nuestro caso DHT22), la dirección del bróker (en nuestro caso en la línea IPAddress(192, 168, 1, 36)), y el SSID/Password de la red WiFi que se utilice. Las instrucciones para el seteo de bibliotecas están en la otra fuente suministrada por la cátedra, que es: <https://randomnerdtutorials.com/esp32-mqtt-publish-dht11-dht22-arduino/>

En el repositorio [1] se han incluido las bibliotecas requeridas, se pueden integrar utilizando el menú Programa (sketch) → Incluir librería → Añadir biblioteca .ZIP (Figura 5)

Elas son: AsyncMqttClient, y AsyncTCP Library

(las Adafruit DHT y Adafruit UnifiedSensor library se pueden instalar desde el IDE directamente, con administrar bibliotecas).

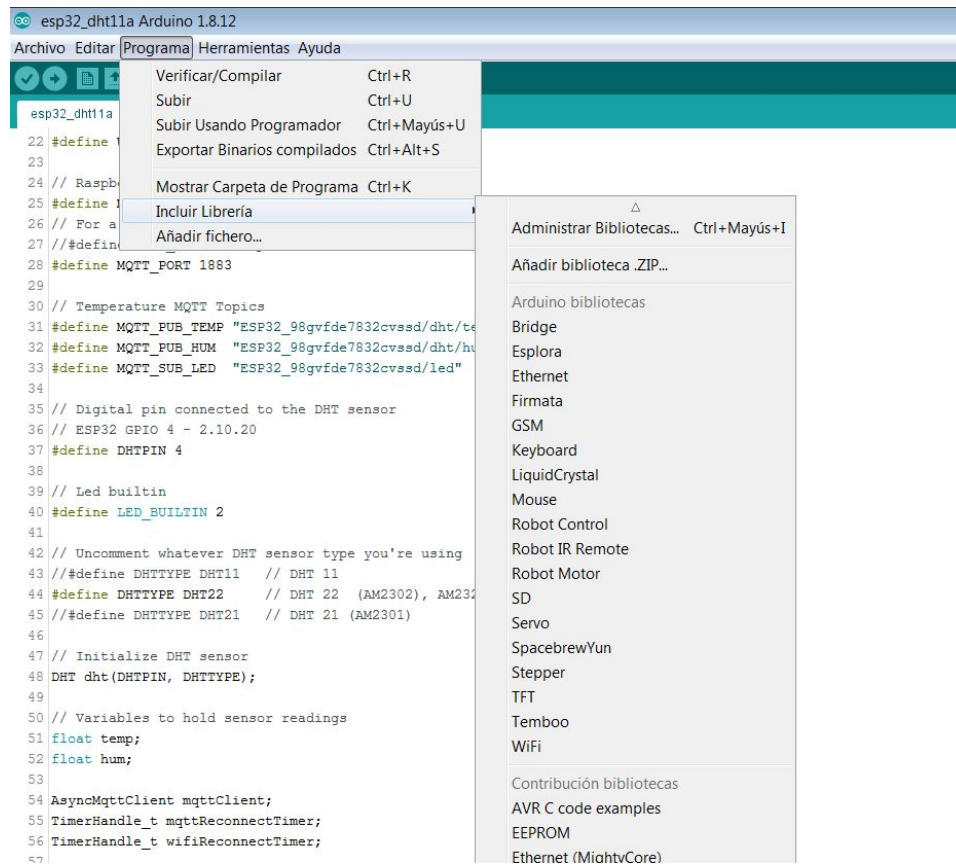


Figura 5 – Inclusión de librerías requeridas para que funcione el .INO de la cátedra.

3. Programacion vía Visual Studio Code + PlatformIO

3.1) Siguiendo la línea utilizada en la materia anterior (Arquitectura de Protocolos) se puede configurar el mismo programa .ino de ESP32 para funcionar como un .cpp con Visual Studio Code + PlatformIO, la estructura es más similar a un IDE de desarrollo convencional en C/C++ y tiene un manejo mas claro de librerías y dependencias.

Aspectos generales de la migración pueden consultarse en la muy buena referencia [2] de randomnerdtutorials. Para este caso particular, se puede elegir entre importar el Arduino Project (mas laborioso) o hacer uno nuevo (Figura 6):

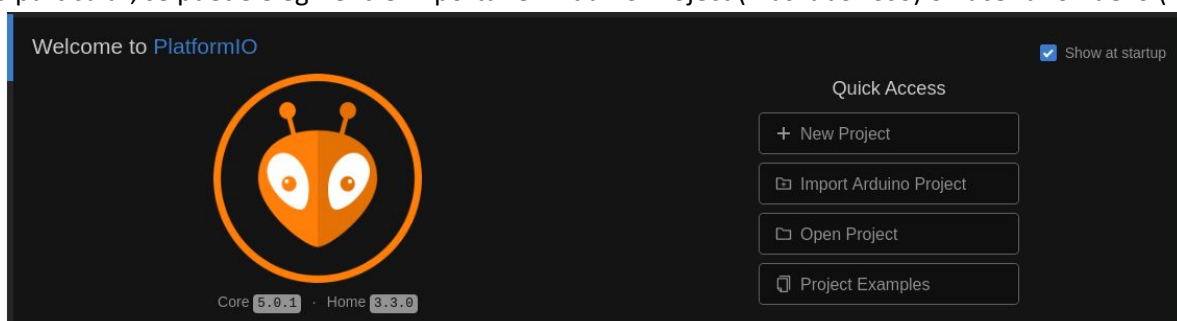


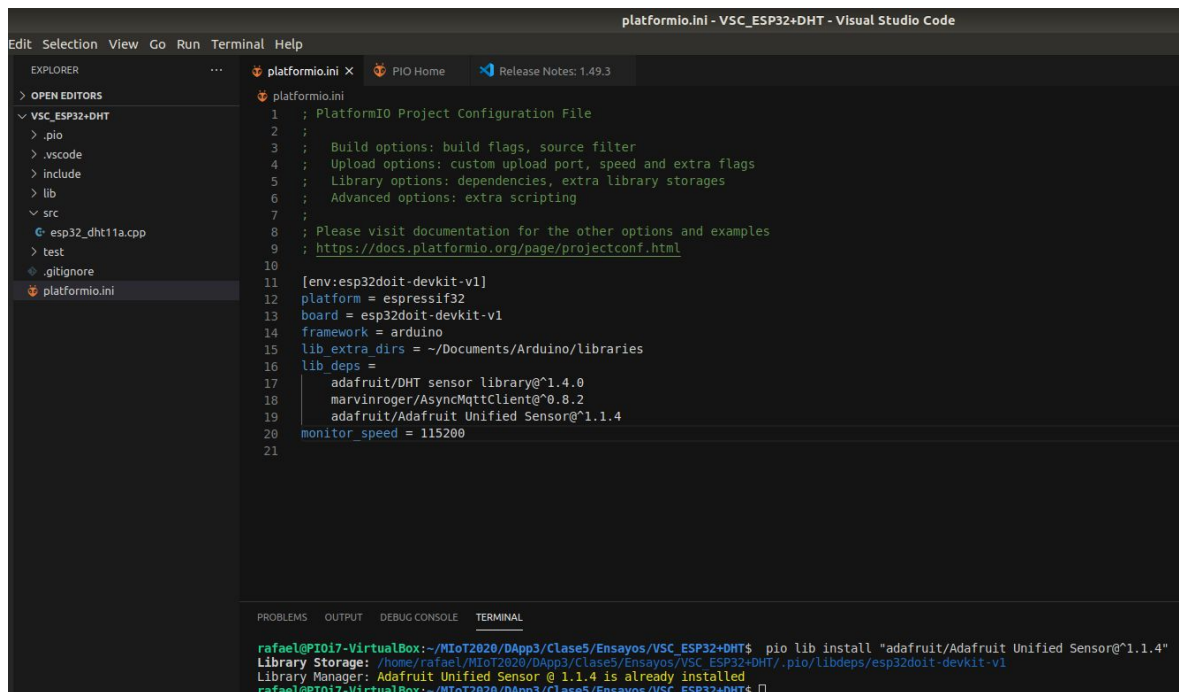
Figura 6 – Opciones al iniciar el PlatformIO desde VSC

Importante recordar que se requiere Python 3.6+ instalado, y además que PIO hace uso de algunos elementos de la Plataforma ArduinoIDE que debe estar instalada en Ubuntu.

Puede ser conveniente verificar con un proyecto muy sencillo Blink las dependencias básicas y la comunicación con la Placa ESP32, recordando que para que programe hay que mantener presionado el botoncito “Boot” luego de mandar Upload (esto con el otro IDE también).

Una vez que está todo operativo, si se crea un **nuevo proyecto en PIO** hay que:

1. Cargar el .ino en un /src/.cpp con el nombre que se crea conveniente. Con Agustín siempre era main.cpp, aquí usamos el mismo nombre del .ino
2. modificar el archivo Platformio.ini como se muestra. Esto depende de la placa que se tenga, con Agustín usábamos siempre la placa [env:alksesp32] pero en este caso se adapta mejor la esp32doit que es la del fabricante de la placa ensayada. Como siempre, agregar al final la velocidad del monitor a 115200, sino sale a 9600:



```
platformio.ini
1 ; PlatformIO Project Configuration File
2 ;
3 ; Build options: build flags, source filter
4 ; Upload options: custom upload port, speed and extra flags
5 ; Library options: dependencies, extra library storages
6 ; Advanced options: extra scripting
7 ;
8 ; Please visit documentation for the other options and examples
9 ; https://docs.platformio.org/page/projectconf.html
10
11 [env:esp32doit-devkit-v1]
12 platform = espressif32
13 board = esp32doit-devkit-v1
14 framework = arduino
15 lib_extra_dirs = ~/Documents/Arduino/libraries
16 lib_deps =
17     adafruit/DHT sensor library@^1.4.0
18     marvinroger/AsyncMqttClient@^0.8.2
19     adafruit/Adafruit Unified Sensor@^1.1.4
20 monitor_speed = 115200
21
```

```
rafael@PT017-VirtualBox:~/MIOT2020/DApp3/Clase5/Ensayos/VSC_ESP32+DHT$ pio lib install "adafruit/Adafruit Unified Sensor@^1.1.4"
Library Storage: /home/rafael/MIOT2020/DApp3/Clase5/Ensayos/VSC_ESP32+DHT/.pio/libdeps/esp32doit-devkit-v1
Library Manager: Adafruit Unified Sensor @ 1.1.4 is already installed
rafael@PT017-VirtualBox:~/MIOT2020/DApp3/Clase5/Ensayos/VSC_ESP32+DHT$
```

Figura 7 – Modificaciones a Platform IO

La inclusión de las bibliotecas requeridas se puede hacer en forma manual, con los siguientes comandos desde un terminal de VSC:

```
pio lib install "adafruit/DHT sensor library@^1.4.0"
```

```
pio lib install "marvinroger/AsyncMqttClient@^0.8.2"
```

```
pio lib install "adafruit/Adafruit Unified Sensor@^1.1.4"
```

3.2) Con lo anterior debería compilar, y recordar modificar a) La selección del pin de datos para DHT (PIN 4 en nuestro caso), la selección del DHT que se tenga (en nuestro caso DHT22), la dirección del bróker (en nuestro caso en la línea IPAddress(192, 168, 1, 36)), y el SSID/Password de la red WiFi que se utilice.

En la Figura 8 se muestra el sistema comunicando a la RBPi, que debe estar encendida y con el Mosquitto funcionando (ver punto 4)

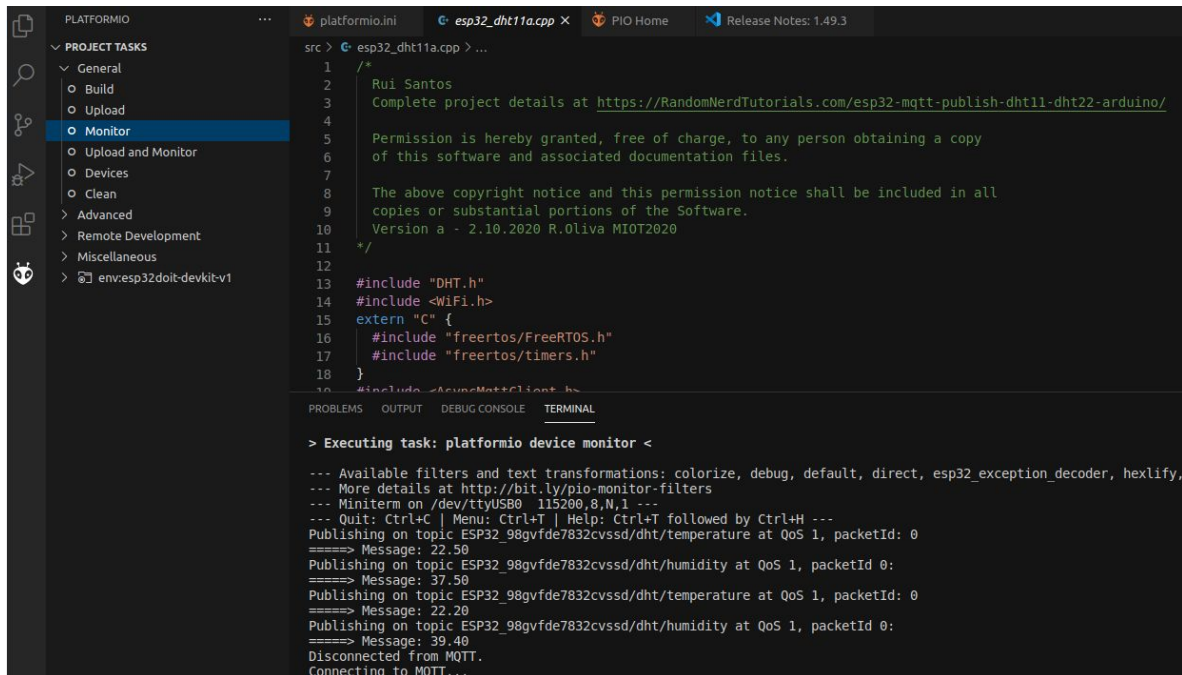


Figura 8 – ESP32 conectada a RBPi desde VSC/PIO

4) Visualización en node-red desde RBPi

4.1) Mosquitto: Se instaló la imagen SD en una RBPi2 dado que la de la cátedra estaba en uso para el Proyecto Final, y funcionó correctamente la vinculación sin modificaciones. Solamente se requirió la instalación del Mosquitto como se indicó en la cátedra con:

```
sudo apt update
sudo apt install -y mosquitto mosquitto-clients
```

Verificar:

```
mosquitto -v (da v1.5.7 en nuestro caso)
```

4.2) Nodos agregados en node-RED: Se agregaron a los tres nodos por defecto (uno de ellos, MyTable, conectado a la base de datos laravel generada con fake names), dos nodos de Mqtt-in (Temp, y Humedad) y dos nodos de Gauges con los mismos nombres. Además dos nodos debug mas abajo que permite ver los datos (figura 9). El flow se ha exportado y está en [1] si se quiere replicar.

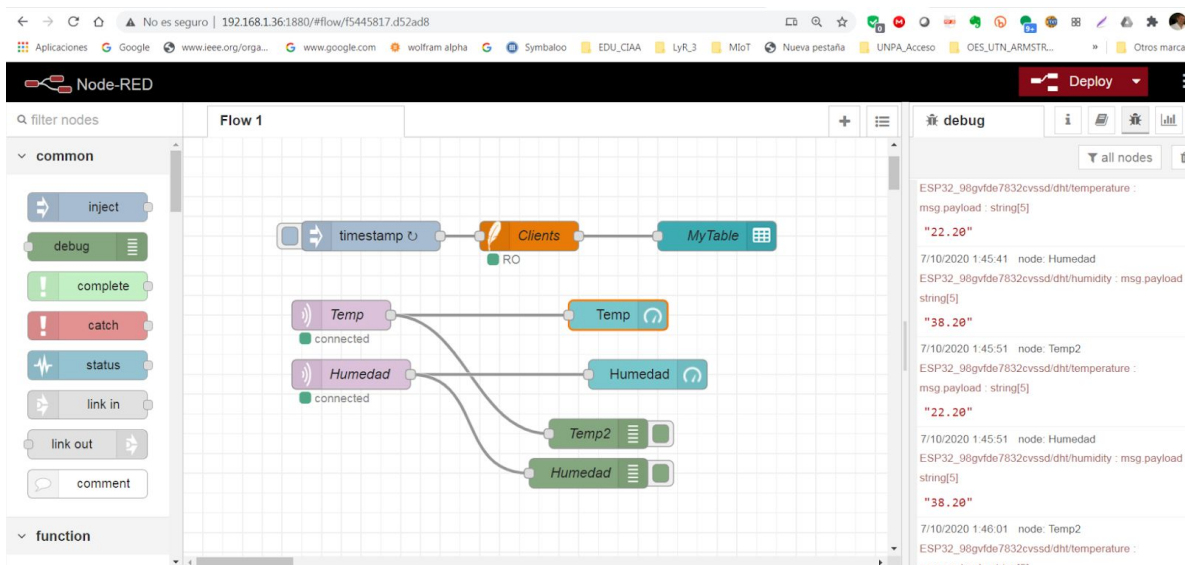


Figura 9 – Node Red con conexión via Mosquitto a la ESP32 desde VSC/PIO

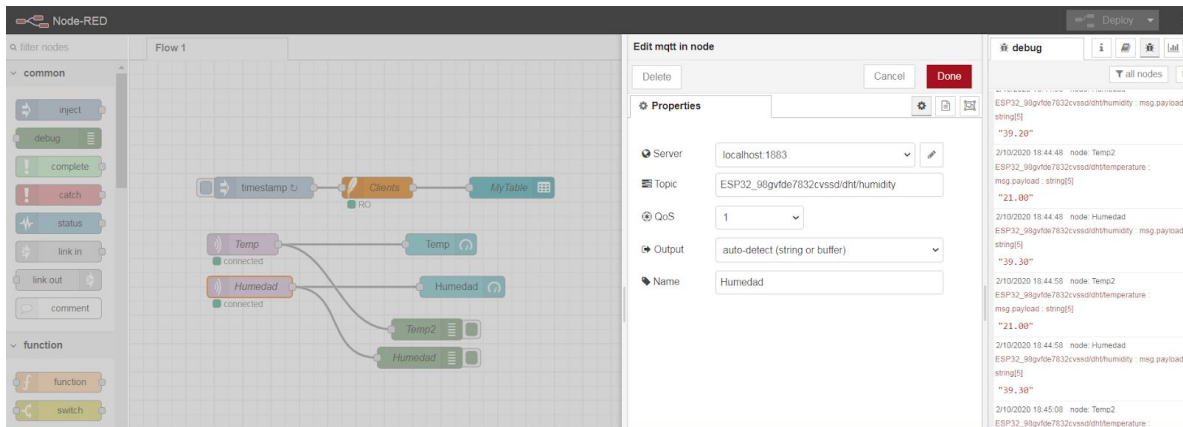


Figura 10 – Detalle config de Mqtt-in para Humedad

4.3) Visualización de los datos node-RED-dashboard: Se puede observar los datos leídos en /ui de la dirección de node-red (dashboard), desde un browser externo en el puerto 1880, como se observa en la Figuras 11 . Todavía no se ha trabajado con la base MyTable que contiene los datos generados en clase y tiene una conexión con este flow.

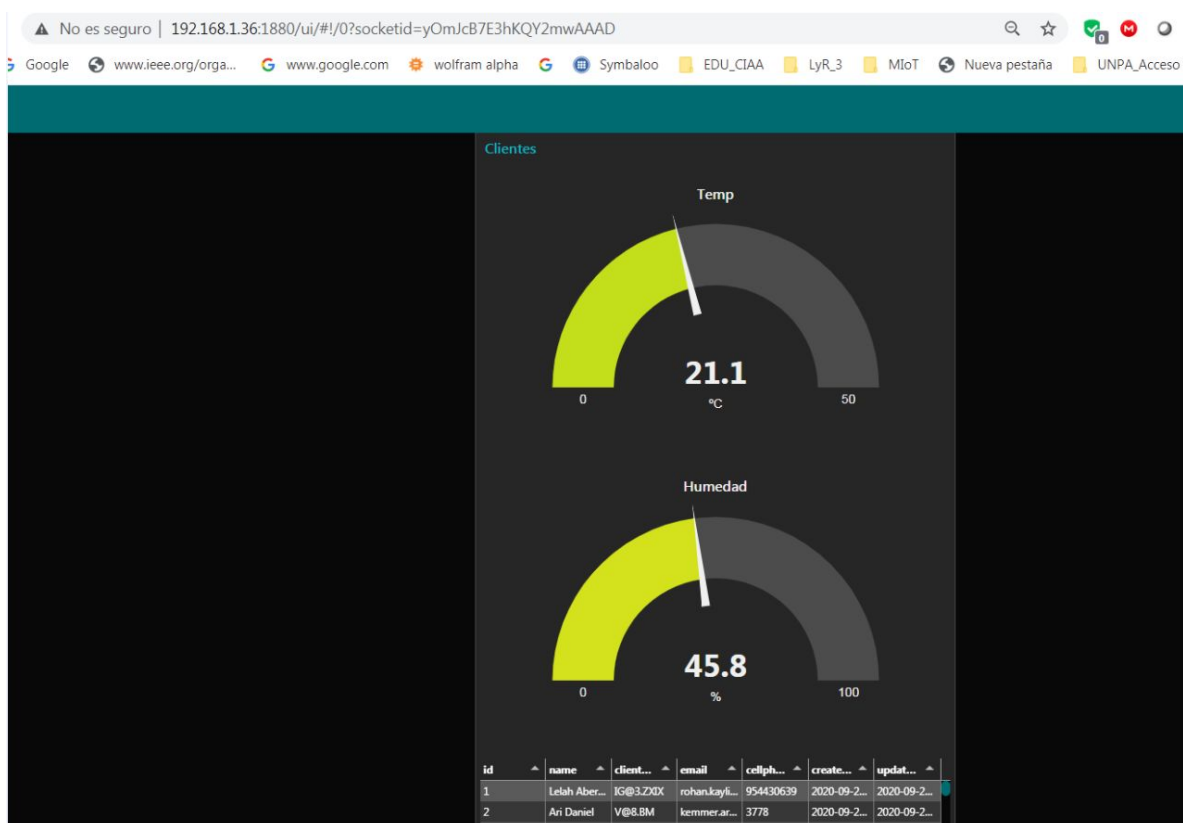


Figura 11 – Visualización de datos via Node-red-dashboard

5). Referencias:

- [1] <https://github.com/rafaeloliva/DapIoT.git>
- [2] <https://randomnerdtutorials.com/vs-code-platformio-ide-esp32-esp8266-arduino/#more-99203>