

Código de VS.Code para el ESP8266

- Librerías Básicas:

```
#include <Arduino.h>
#include <ESP8266WiFi.h>
#include <ArduinoJson.h>
```

- Librerías HTTP y MQTT:

```
#include <ArduinoHttpClient.h>
#include <PubSubClient.h>
```

- Librerías Sensor DHT11:

```
#include "DHT.h"
#include <Adafruit_Sensor.h>
#include <DHT_U.h>
```

- LibreríaS Sensor GPS:

```
#include <TinyGPS++.h>
#include <SoftwareSerial.h>
```

- Librería Ticker:

```
#include <Ticker.h>
```

- Librerías Altavoz:

```
#include "AudioFileSourcePROGMEM.h"
#include "AudioGeneratorWAV.h"
#include "AudioOutputI2SNoDAC.h"
#include "viola.h"
```

- Librería Servos:

```
#include <Servo.h>
```

```
////////////////////////////////// VARIABLES GENERALES //////////////////////////////////
```

```
// variables para la página html
```

```
String temperatura_;
```

```
String humedad_;
```

```
String nivel_gas_;
```

- Tres variables para las distintas mediciones de los sensores, que usaremos más adelante, en la función auxiliar ventana_open() y en el loop()

```
////////////////////////////////// SENSOR GPS //////////////////////////////////
```

```
static const int RXPin = 3, TXPin = 1; static const uint32_t GPSBaud = 9600;
```

```
TinyGPSPlus gps;
```

```
SoftwareSerial gpsSerial(RXPin, TXPin);
```

```
char buffer[100];
```

```
int idtipo_gps = 490; String tipo_gps = "gps"; String nombre_gps = "NEO-M6"; int idsensor_gps = 490; //variables para los metodos REST  
float latitude , longitude; String lat_str , lng_str; // variables para la funcion printData()
```

- Variables para las funciones displayInfo() y printData() usadas para la librería TinyGPS++ y SoftwareSerial.
- Variables para usar en los métodos REST para la base de datos:
`idtipo_gps`, `tipo_gps`, `nombre_gps`, `idsensor_gps`.

```
////////////////////////////////// SENSOR DHT_11 //////////////////////////////////
```

```
#define DHTPIN D1
```

```
#define DHTTYPE DHT11
```

```
DHT dht(DHTPIN, DHTTYPE);
```

```
int id_dht11 = 12; String tipo_dht11 = "humedad_temperatura"; String nombre_dht11 = "DHT_11"; // variables para los metodos REST
```

- Variables para la librería DHT.h: `DHTPIN`, `DHTTYPE` y `DHT`
- Variables para usar en los métodos REST para la base de datos:
`id_dht11`, `tipo_dht11` y `nombre_dht11`

```
////////////////////////////////// SENSOR MQ-2 //////////////////////////////////
```

```
int id_mq2 = 2; String tipo_mq2 = "humo"; String nombre_mq2 = "MQ_2";  
// variables para los metodos REST
```

- Variables para usar en los métodos REST para la base de datos:
`id_mq2` , `tipo_mq2` y `nombre_mq2`

```

////////////////////////////////// ACTUADOR ALTAVOZ //////////////////////////////////
AudioGeneratorWAV *wav;
AudioFileSourcePROGMEM *file;
AudioOutputI2SNoDAC *out;

```

- Variables para generar sonido en el altavoz, definidas en la librería ESP8266Audio

```

////////////////////////////////// ACTUADOR SERVO //////////////////////////////////
Servo myservo1;
Servo myservo2;

WiFiServer server(80);

```

- Definiciones de los dos servos usados en el proyecto, de tipo `Servo`
- Declaración del servidor del ESP por el puerto 80 `WiFiServer` (incluidas en las librerías de arduino)

```

////////////////////////////////// CONEXIONES //////////////////////////////////
IPAddress serverAddress(192, 168, 0, 16); // ip del pc

//variables para conexión ESP8266/wifi
WiFiClient wifi_http; int port_http = 8080;
HttpClient client_http = HttpClient(wifi_http, serverAddress,
port_http); //cliente HTTP

//variables para conexión MQTT
WiFiClient wifi_mqtt; int port_mqtt = 1883; const char usser_pass[7] =
"admin";
PubSubClient client_mqtt(serverAddress,port_mqtt,wifi_mqtt); //cliente
MQTT

long last_msg = 0; char msg_mqtt[100];

```

- Variables para las conexiones tanto HTTP como MQTT:
 - `IPAddress serverAddress`: dirección IP del Servidor, en este caso mi PC
 - `WiFiClient` para definir el tipo de cliente, que será `HttpClient` y `PubSubClient`
 - `port_http` y `port_mqtt` para definir los puertos para HTTP y MQTT

```
////////////////////////////////// FUNCIONES AUXILIARES //////////////////////////////////
```

```
//FUNCIÓN AUXILIAR PARA EL SERVO
void ventana_open(float temp, float hum, float gas){ // funcion para
comprobar los niveles de temperatura, humedad y gas para activar el
servo
    if(temp > 24 || hum > 90 || gas > 60){
        for (int angulo = 0; angulo <= 90; angulo++){ // se activa el
servo a 90 grados (ventana)
            myservo1.write(angulo);
            myservo2.write(angulo);
            delay(10);
        }
    }
    if(temp>0){
        temperatura_ = temp;
    }
    if(hum>0){
        humedad_ = hum;
    }
    if(gas>0){
        nivel_gas_ = gas;
    }
}
```

- Esta función auxiliar tiene dos cometidos, el principal es la condición de que si uno de los parámetros recibidos supera cierto umbral, activamos los servos (en este caso ventanas) de tal forma que cada vez que realicemos lecturas de los sensores de temperatura/humedad o nivel de gas **siempre** llamamos a esta función.
- La segunda funcionalidad que tiene, debido a que siempre hay que llamar a esta función, es el lugar idóneo para usar las variables de tipo String: `temperatura_`, `humedad_` y `nivel_gas_` que posteriormente se usarán en la página HTML para mantener la información de los sensores actualizadas.

```

////////////////////////////////// CONEXIÓN WIFI ////////////////////////////////////
void setup_wifi(){
  WiFi.begin("vodafone7638", "N5ZXFUJGH5AK4Y");
  Serial.print("\nConectando Wifi:");
  WiFi.mode(WIFI_STA);
  while (WiFi.status() != WL_CONNECTED)
  {
    delay(50);
  }
  Serial.print(" --> Wifi Conectado: IP address -> ");
  Serial.print(WiFi.localIP());
}

```

- Función auxiliar que llamaremos en el método setup, para iniciar la conexión wifi del ESP8266

```

////////////////////////////////// CONEXIÓN MQTT ////////////////////////////////////
void mqtt_setup(){
  Serial.print("\n\nMQTT Iniciado:");
  delay(10);
  pinMode(LED_BUILTIN,OUTPUT);
  client_mqtt.setServer(serverAddress,port_mqtt);
  client_mqtt.setCallback(callback);
  mqtt_reconnect();
}

```

- Función auxiliar que llamaremos en el método setup, iniciamos el servidor MQTT con **setServer** y añadimos la función de callback predefinida con **setCallback**
- Al final de la función llamamos directamente a la función **mqtt_reconnect()**

```

////////////////////////////////// CONEXIÓN MQTT ////////////////////////////////////
void mqtt_loop() {
  if (!client_mqtt.connected()) {
    mqtt_reconnect();
  }
  client_mqtt.loop();
  long now = millis();
  if (now - last_msg > 2000) {
    last_msg = now;
    Serial.println(msg_mqtt);
  }
}

```

- Función auxiliar que llamaremos continuamente en el método loop, comprueba si el cliente está conectado correctamente y supervisa la llegada de mensajes.

```

////////////////////////////////// CONEXIÓN MQTT //////////////////////////////////
void mqtt_reconnect() {
    String client_Id = "ESP8266Client"; //nombre de cliente
    client_Id += String(random(0xffff), HEX);
    while (!client_mqtt.connected()) {
        Serial.print("\nEsperando a la Conexion MQTT...");
        if (client_mqtt.connect(client_Id.c_str(),usser_pass,usser_pass)) {
            Serial.print("\nMQTT Conectado --> ");
            Serial.print("ID Cliente --> ");
            Serial.print(client_Id);
            client_mqtt.subscribe("control_puerta");
            client_mqtt.subscribe("control_ventana");
            client_mqtt.subscribe("control_sensores");
        } else {
            Serial.print("\nFailed, rc=");
            Serial.print(client_mqtt.state());
            Serial.println("Try again in 5 seconds: ");
            for(int i=5;i>0;i--){
                delay(1000);
                Serial.print(i);
                Serial.print(", ");
            }
        }
    }
}

```

- Esta función crea un id_cliente random para el ESP8266 y pide la conexión con él.
- Si consigue conectarse, se suscribe a **control_puerta**, **control_ventana** y **control_sensores** con el método **subscribe**, que serán los mensajes topic usados en el proyecto.

```

////////////////////////////////// CONEXIÓN MQTT ////////////////////////////////////

void callback(char* topic, byte* payload, unsigned int length) {
    digitalWrite(LED_BUILTIN, LOW);

    Serial.print("\nMessage arrived [");
    Serial.print(topic);
    Serial.print("]\n ");

    Serial.print("Message: ");
    for (int i = 0; i < length; i++) {
        Serial.print((char)payload[i]);
    }

    String topicStr(topic);

```

En esta primera parte, mostramos por consola el topic MQTT recibido y lo copiamos en la variable `topicStr`

```

    if (topicStr.compareTo("control_puerta")==0){ //si el topic es
control_puerta

        if ((char)payload[0] == '1') { //1 para entrar al bus, se activa el
servo a 180 grados (puerta)
            //SERVOS ON
            for (int angulo = 0; angulo <= 180; angulo++){
                myservo1.write(angulo);
                myservo2.write(180-angulo);
                delay(20);
            }
            delay(3000);
            for (int angulo = 180; angulo>= 0; angulo--){
                myservo1.write(angulo);
                myservo2.write(180-angulo);
                delay(20);
            }
        }

        if ((char)payload[0] == '0') { //0 para salir del bus, se activa el
servo a 180 grados (puerta) y suena el altavoz para indicar al
conductor que alguien se quiere bajar
            //ALTAVOZ ON
            audioLogger = &Serial;
            file = new AudioFileSourcePROGMEM( viola, sizeof(viola) );
            out = new AudioOutputI2SNoDAC();
            wav = new AudioGeneratorWAV();

```

```

wav->begin(file, out);

//SERVOS ON
for (int angulo = 0; angulo <= 180; angulo++){
    myservo1.write(angulo);
    myservo2.write(180-angulo);
    delay(20);
}
delay(3000);
for (int angulo = 180; angulo >= 0; angulo--){
    myservo1.write(angulo);
    myservo2.write(180-angulo);
    delay(20);
}
}
}

```

Primer posible topic → `control_puerta`: (servos actúan como una puerta)

- si el mensaje contiene un 1, quiere decir que alguien se quiere subir al autobús, por lo que activamos los servos 1 y 2 a 180 grados
- si el mensaje contiene un 0, quiere decir que alguien se quiere bajar del autobús, por lo que activamos los servos 1 y 2 a 180 grados y activamos el altavoz, con el audio introducido, en este caso: `viola.h`

```

if (topicStr.compareTo("control_ventana")==0) { //si el topic es
control_ventana

    if ((char)payload[0] == '1') { //1 para servo D3, se activa el
servo a 90 grados (ventana n°1)
        //SERVO D3 ON
        for (int angulo = 0; angulo <= 90; angulo++){
            myservo1.write(angulo);
            delay(10);
        }
    }
    if ((char)payload[0] == '0') { //0 para servo D0, se activa el
servo a 90 grados (ventana n°2)
        //SERVO D0 ON
        for (int angulo = 0; angulo <= 90; angulo++){
            myservo2.write(angulo);
            delay(10);
        }
    }
}
}

```


Segundo posible topic → `control_ventana`: (servos actúan como una ventana)

- si el mensaje contiene un 1, activamos el servo1, colocado en el pin D3
- si el mensaje contiene un 0, activamos el servo2, colocado en el pin D0

```
if (topicStr.compareTo("control_sensores")==0){ //si el topic es
control_sensores

    if ((char)payload[0] == '1') { //1 para para sensor temperatura

        float h = dht.readHumidity();
        float t = dht.readTemperature(); // or dht.readTemperature(true)
for Fahrenheit

        if (isnan(h) || isnan(t)) {
            Serial.println("Failed to read from DHT sensor!");
        }
        ventana_open(t,h,0);
        Serial.print(("  Temperature: "));
        Serial.print(t);
        Serial.print((" grados ||"));
        Serial.print(("  Humededity: "));
        Serial.print(h);
        Serial.print("% ");
    }
    if ((char)payload[0] == '0') { //0 para sensor gas
        Serial.print("\n\nMQ-2 test -> ");
        float h = analogRead(A0);
        if(isnan(h)){
            Serial.println("ERROR NO DETECTA SENSOR MQ-2");
        }
        ventana_open(0,0,h/1023*100);
        Serial.print("Nivel de gas: ");
        Serial.print(h/1023*100);
    }
}

digitalWrite(LED_BUILTIN, HIGH); // Turn the LED off by making the
voltage HIGH
}
```

Tercer posible topic → `control_sensores`:

- si el mensaje contiene un 1, activamos el sensor DTH11
- si el mensaje contiene un 0, activamos el sensor MQ-2

Lo último que cabe mencionar de esta función, es que activamos el led del ESP

`digitalWrite(LED_BUILTIN, LOW);` cuando llega un mensaje, y desactivamos el led con `digitalWrite(LED_BUILTIN, HIGH);` al final de la función.

Funciones tipo serialize para las clases de la base de datos, en el proyecto hacemos uso de tres:

serialize_GPS_Info, para la información asociada al GPS

serialize_Sensor_Info, para la información de los distintos sensores

serialize_Sensor_Data, para los datos recogidos de los distintos sensores

```
String serialize_GPS_Info(int idtipo_gps, float last_value_x, float
last_value_y, int idsensor){
    StaticJsonDocument<200> doc;
    doc["idtipo_gps"] = idtipo_gps;
    doc["x"] = last_value_x;
    doc["y"] = last_value_y;
    doc["idsensor"] = idsensor;
    String output;
    serializeJson(doc,output);
    return output;
}

String serialize_Sensor_Info(int idsensor, String tipo, String nombre,
float last_value1, float last_value2, int iddispositivo){
    StaticJsonDocument<200> doc;
    doc["idsensor"] = idsensor;
    doc["tipo"] = tipo;
    doc["nombre"] = nombre;
    doc["last_value1"] = last_value1;
    doc["last_value2"] = last_value2;
    doc["iddispositivo"] = iddispositivo;
    String output;
    serializeJson(doc,output);
    return output;
}

String serialize_Sensor_Data(String timestamp,float valor1, float
valor2, int idsensor){
    DynamicJsonDocument doc(200);
    doc["timestamp"] = timestamp;
    doc["valor1"] = valor1;
    doc["valor2"] = valor2;
    doc["idsensor"] = idsensor;
    String output;
    serializeJson(doc,output);
    return output;
}
```

Funciones auxiliares para los sensores:

- Dos funciones para cada sensor, una para el método Setup() y otra para el método Loop()

- SENSOR GPS:

- Función `void init_GPS()` para el método setup, que realiza un POST en la clase Info_Sensor y otro POST en la clase Tipo_GPS con valores iniciales x e y igual a 0
- Función `void sensor_GPS()` para el método loop, que realiza un POST en la clase Data_Sensor y otros dos PUT en la clase Tipo_GPS y Info_Sensor con valores iniciales x e y igual a un valor random

```
void init_GPS(){ //INICIALIZACIÓN EN SETUP: introduzco con un post del
sensor gps correspondiente a la base de datos --> Info_Sensor y en
Tipo_GPS
    String contentType = "application/json";
    //POST EN INFO_SENSOR
    Serial.print("\n\n -- POST_1 GPS (setup) --> INFO_SENSOR --");
    String body_info_sensor = serialize_Sensor_Info(idsensor_gps,
tipo_gps, nombre_gps, 00.00, 00.00, 1);
```

Variable de tipo String `body_info_sensor` que hace uso de la función `serialize`

```
    client_http.beginRequest();
    client_http.post("/api/Post_Info_Sensor/", contentType,
body_info_sensor.c_str());
    int code_gps_info = client_http.responseStatusCode();
    if(code_gps_info==200 || code_gps_info==201){
        Serial.print("\nCode: ");
        Serial.print(code_gps_info);
        Serial.print("\nBody: ");
        Serial.print(client_http.responseBody());
    }else if(code_gps_info == 401){
        Serial.print("\nCode: ");
        Serial.print(code_gps_info);
        Serial.print(" --> Sensor ya instalado");
    }else{
        Serial.print("\nCódigo de error: ");
        Serial.print(code_gps_info);
    }
    client_http.endRequest();
```

`beginRequest()` y `endRequest()` para hacer uso de varias llamadas a los métodos REST y dependiendo de si el código del método es 200 o 201, imprime todo correctamente, si es 401, es que el recurso ya existe y cualquier otro código de error, que lo muestre por pantalla

```

//POST Tipo_gps --> idtipo_gps / x / y / idsensor
Serial.print("\n\n  --  POST_2 GPS (setup) --> TIPO_GPS --");
String body_info_gps = serialize_GPS_Info(idtipo_gps, 00.00, 00.00,
idsensor_gps);

client_http.beginRequest();
client_http.post("/api/PostGPS/", contentType,
body_info_gps.c_str());
int code_gps = client_http.responseStatusCode();
if(code_gps==200 || code_gps==201){
    Serial.print("\nCode: ");
    Serial.print(code_gps);
    Serial.print("\nBody: ");
    Serial.print(client_http.responseBody());
}else if(code_gps == 401){
    Serial.print("\nCode: ");
    Serial.print(code_gps);
    Serial.print(" --> Sensor ya instalado");
}else{
    Serial.print("\nCódigo de error: ");
    Serial.print(code_gps);
}
client_http.endRequest();
}

```

```

void sensor_GPS(){ //FUNCION EN LOOP: para actualizar los valores del
sensor gps con un POST en Data_Sensor y dos PUTS en Info_Sensor y
Tipo_gps

    float x = random(100000);
    float y = random(100000);

    //post datasensor --> timestamp / valor_X / valor_Y / idsensor
    Serial.println("\n POST PERIÓDICO GPS --> DATA_SENSOR:");
    String contentType = "application/json";
    String body_data_sensor = serialize_Sensor_Data("null", x, y,
idsensor_gps);

    client_http.beginRequest();
    client_http.post("/api/Post_Data_Sensor/", contentType,
body_data_sensor.c_str());
}

```

```

int code = client_http.responseStatusCode();
if(code==200 || code==201){
    Serial.print("\nCode: ");
    Serial.print(code);
    Serial.print("\nBody: ");
    Serial.print(client_http.responseBody());
}else if(code == 401){
    Serial.print("\nCode: ");
    Serial.print(code);
    Serial.print(" --> Fecha ya existente");
}else{
    Serial.print("\nCódigo de error: ");
    Serial.print(code);
}
client_http.endRequest();

//PUT Info_Sensor ---> idsensor / tipo / nombre / last_value1
/ last_value2 / iddispositivo
Serial.println("\n PUT PERIÓDICO GPS --> INFO_SENSOR:");
String body_info_sensor = serialize_Sensor_Info(idsensor_gps,
tipo_gps, nombre_gps, x, y, 1);

client_http.beginRequest();

client_http.put("/api/PutInfoSensor/490",contentType,body_info_sensor.c
_str());
Serial.print("\nCode: ");
Serial.print(client_http.responseStatusCode());
Serial.print("\nBody: ");
Serial.print(client_http.responseBody());
client_http.endRequest();

//PUT Tipo_gps ---> idtipo_gps / x / y / idsensor
Serial.print("\n\n -- PUT PERIÓDICO GPS --> TIPO_GPS --");
String body_info_gps = serialize_GPS_Info(idtipo_gps, x, y,
idsensor_gps);

client_http.beginRequest();
client_http.put("/api/PutSensorGPS/490", contentType,
body_info_gps.c_str());
Serial.print("\nCode: ");

```

```
Serial.print(client_http.responseStatusCode());  
Serial.print("\nBody: ");  
Serial.print(client_http.responseBody());  
client_http.endRequest();  
}
```

Para el GPS, hay dos funciones auxiliares más no usadas, debido al funcionamiento “incorrecto” del sensor: `displayInfo()` y `printData()`

A la hora de tomar los datos del satélite, he visto en Internet que puede pasar que el sensor requiere de un tiempo para completar todos los datos necesarios, inicialmente los datos están encriptados con „999,,,,,23,,,,, ... y es cierto que algunos de ellos los ha recogido el sensor en un tiempo de más de una hora, pero debido a que estoy con el PC en un sótano y que no es lo principal en el proyecto, hemos decidido centrarnos en otras cosas más importantes

- **SENSOR DTH_11:**

- Función `void init_DTH_11()` para el método setup, que realiza un POST en la clase Info_Sensor
- Función `void sensor_DHT11()` para el método loop, que realiza un POST en la clase Data_Sensor y otro PUT en la clase Info_Sensor con los valores tomados del sensor

```
//SENSOR DHT11
void init_DTH_11(){ //INICIALIZACIÓN EN SETUP: introduzco con un post
el sensor correspondiente a la base de datos --> Info_Sensor

    Serial.print("\n\n  --  POST DHT_11 (setup) --> INFO_SENSOR --");

    String contentType = "application/json";
    dht.begin();
    String body_info = serialize_Sensor_Info(id_dht11, tipo_dht11,
nombre_dht11, 00.00, 00.00, 1);

    client_http.beginRequest();
    client_http.post("/api/Post_Info_Sensor/", contentType,
body_info.c_str());
    int code_dht = client_http.responseStatusCode();
    if(code_dht==200 || code_dht==201){
        Serial.print("\nCode: ");
        Serial.print(code_dht);
        Serial.print("\nBody: ");
        Serial.print(client_http.responseBody());
    }else if(code_dht == 401){
        Serial.print("\nCode: ");
        Serial.print(code_dht);
        Serial.print(" --> Sensor ya instalado");
    }else{
        Serial.print("\nCódigo de error: ");
        Serial.print(code_dht);
    }
    client_http.endRequest();
}
```

```

void sensor_DHT11(){ //FUNCION EN LOOP: para actualizar los valores del
sensor con un put en Info_Sensor y un post en Data_Sensor

    //lectura de temperatura y humedad:
    Serial.print("\nDHT11 test --> ");
    float h = dht.readHumidity();
    float t = dht.readTemperature(); // or dht.readTemperature(true) for
Fahrenheit
    if (isnan(h) || isnan(t)) {
        Serial.println("Failed to read from DHT sensor!");
        return;
    }
}

```

- Lectura de datos del sensor

```

ventana_open(t,h,0);

```

- Llamada a la función ventana_open()

```

Serial.print("  Temperature: ");
Serial.print(t);
Serial.print(" grados ||");
delay(1500);
Serial.print("  Humededity: ");
Serial.print(h);
Serial.print("% ");
delay(1500);

```

- Mostrar datos por consola

```

//métodos rest
String contentType = "application/json";

//put infosensor --> idsensor / tipo / nombre / last_value1 /
last_value2 / iddispositivo
Serial.print("\n\n  -- PUT PERIÓDICO DHT_11 --> INFO_SENSOR --");
String body_info = serialize_Sensor_Info(id_dht11, tipo_dht11,
nombre_dth11, t, h, 1);
client_http.beginRequest();

client_http.put("/api/PutInfoSensor/12",contentType,body_info.c_str());
Serial.print("\nCode: ");
Serial.print(client_http.responseStatusCode());
Serial.print("\nBody: ");
Serial.print(client_http.responseBody());
client_http.endRequest();

delay(1500);

```



```
//post datasensor --> timestamp / valor1 / valor2 / idsensor

Serial.print("\n\n  -- POST PERIÓDICO DHT_11 --> DATA_SENSOR --");
String body_data = serialize_Sensor_Data("null", t, h, id_dht11);

client_http.beginRequest();
client_http.post("/api/Post_Data_Sensor/", contentType, body_data);
int code = client_http.responseStatusCode();
if(code==200 || code==201){
    Serial.print("\nCode: ");
    Serial.print(code);
    Serial.print("\nBody: ");
    Serial.print(client_http.responseBody());
}else if(code == 401){
    Serial.print("\nCode: ");
    Serial.print(code);
    Serial.print(" --> Fecha ya existente");
}else{
    Serial.print("\nCódigo de error: ");
    Serial.print(code);
}
client_http.endRequest();
}
```

- **SENSOR MQ-2:**

- Función `void init_MQ-2()` para el método setup, que realiza un POST en la clase Info_Sensor
- Función `void sensor_MQ-2()` para el método loop, que realiza un POST en la clase Data_Sensor y otro PUT en la clase Info_Sensor con los valores tomados del sensor

De igual forma que para el sensor DTH_11:

```
//SENSOR MQ2
void init_MQ_2(){ //INICIALIZACIÓN EN SETUP: introduzco con un post el
sensor correspondiente a la base de datos --> Info_Sensor

    Serial.print("\n\n  --  POST MQ-2 (setup) --> INFO_SENSOR --");

    String contentType = "application/json";
    String body_info = serialize_Sensor_Info(id_mq2, tipo_mq2,
nombre_mq2, 00.00, 00.00, 1);

    client_http.beginRequest();
    client_http.post("/api/Post_Info_Sensor/", contentType,
body_info.c_str());
    int code_mq2 = client_http.responseStatusCode();
    if(code_mq2==200 || code_mq2==201){
        Serial.print("\nCode: ");
        Serial.print(code_mq2);
        Serial.print("\nBody: ");
        Serial.print(client_http.responseBody());
    }else if(code_mq2 == 401){
        Serial.print("\nCode: ");
        Serial.print(code_mq2);
        Serial.print(" --> Sensor ya instalado");
    }else{
        Serial.print("\nCódigo de error: ");
        Serial.print(code_mq2);
    }
    client_http.endRequest();
}
```

```
void sensor_MQ_2() { //FUNCION EN LOOP: para actualizar los valores del
sensor con un put en Info_Sensor y un post en Data_Sensor
```

```
    //lectura mq-2
    Serial.print("\n\nMQ-2 test -> ");
    float h = analogRead(A0);
    if(isnan(h)) {
        Serial.println("ERROR NO DETECTA SENSOR MQ-2");
        return;
    }
}
```

- Lectura de datos del sensor

```
ventana_open(0,0,h/1023*100);
```

- Llamada a la función ventana_open()

```
Serial.print("Nivel de gas: ");
Serial.print(h/1023*100);
delay(1500);
```

- Mostrar datos por consola

```
//métodos rest
String contentType = "application/json";

//put infosensor --> idsensor / tipo / nombre / last_value1 /
last_value2 / iddispositivo
Serial.print("\n\n -- PUT PERIÓDICO MQ-2 --> INFO_SENSOR --");
String body_info = serialize_Sensor_Info(id_mq2, tipo_mq2,
nombre_mq2, h/1023*100, 00.00, 1);

client_http.beginRequest();
client_http.put("/api/PutInfoSensor/2", contentType, body_info);
Serial.print("\nCode: ");
Serial.print(client_http.responseStatusCode());
Serial.print("\nBody: ");
Serial.print(client_http.responseBody());
client_http.endRequest();

delay(1500);

//post datasensor --> timestamp / valor1 / valor2 / idsensor

Serial.print("\n\n -- POST PERIÓDICO MQ-2 --> DATA_SENSOR --");
String body_data = serialize_Sensor_Data("null", h/1023*100, 00.00,
id_mq2);

client_http.beginRequest();
```

```
client_http.post("/api/Post_Data_Sensor/", contentType, body_data);
int code = client_http.responseStatusCode();
if(code==200 || code==201){
    Serial.print("\nCode: ");
    Serial.print(code);
    Serial.print("\nBody: ");
    Serial.print(client_http.responseBody());
}else if(code == 401){
    Serial.print("\nCode: ");
    Serial.print(code);
    Serial.print(" --> Fecha ya existente");
}else{
    Serial.print("\nCódigo de error: ");
    Serial.print(code);
}
client_http.endRequest();
}
```

```

////////////////////////////////// TICKERS ////////////////////////////////////
Ticker timer_gps(sensor_GPS,60000); //1 MIN
Ticker timer_dht11(sensor_DHT11, 240000); // 4 MIN
Ticker timer_mq2(sensor_MQ_2, 120000); //2 MIN

```

Defino los **tickers** antes de la función Setup para poder reproducir las funciones del loop cada cierto tiempo periódico

```

////////////////////////////////// SETUP ////////////////////////////////////
void setup() {
    delay(3000);
    Serial.begin(9600);
    Serial.println("\n\nDispositivo arrancado.");

    //INICIO CONEXIÓN WIFI
    setup_wifi();

```

- Llamada a la función setup_wifi()

```

//INICIO CONEXIÓN MQTT
mqtt_setup();

```

- Llamada a la función mqtt_setup()

```

//SERVIDOR HTTP PARA ESP8266NODE
server.begin(); //Iniciamos el servidor
Serial.print("\n\nServidor ESP Iniciado -->");
Serial.print("Ingrese desde un navegador web usando la siguiente IP
--> ");
Serial.print(WiFi.localIP()); //Obtenemos la IP

```

- Inicializamos el servidor HTML para el ESP8266 → `server.begin();`

```

//INICIALIZACIÓN GPS --> POST1: idsensor / tipo / nombre /
last_value1 / last_value2 / iddispositivo || POST2: idtipo_gps /
x / y / idsensor
init_GPS();
//gpsSerial.begin(GPSBaud);

//INICIALIZACIÓN DE DHT11 --> POST: idsensor / tipo / nombre /
last_value1 / last_value2 / iddispositivo
init_DTH_11();

//INICIALIZACIÓN DE MQ-2 --> POST: idsensor / tipo / nombre /
last_value1 / last_value2 / iddispositivo
init_MQ_2();

```

- Llamada a las funciones del setup para los sensores, para añadirlos inicialmente a la base de datos (métodos POST)

```
//INICIALIZACIÓN DE ALTAVOZ
audioLogger = &Serial;
file = new AudioFileSourcePROGMEM( viola, sizeof(viola) );
out = new AudioOutputI2SNoDAC();
wav = new AudioGeneratorWAV();
//wav->begin(file, out);
```

- Inicialización del altavoz, para reproducir el sonido hacemos uso del wav->begin

```
//INICIALIZACION SERVO
myservo1.attach(D2);
myservo2.attach(D0);
```

- Inicializamos los servos a los pines correspondientes del ESP8266

```
//Start tickers
timer_gps.start();
timer_dht11.start();
timer_mq2.start();
}
```

- Método **.start()** para iniciar los tickers(timers) correspondientes para la función loop

```
void loop() {
  mqtt_loop();
```

- Supervisión de mensajes y de conexión MQTT con mqtt_loop()

```
//COMPROBACIÓN DE ALTAVOZ
```

```
if (wav->isRunning()) {
  if (!wav->loop()) wav->stop();
}
```

- Debido a que no hacemos uso de interrupciones, comprobación de si el altavoz está reproduciendo un audio (esto se debe a que si el altavoz está sonando, puede ser interrumpido por ejemplo por otra función más bloqueante, como puede ser

```
timer_gps.update())
```

```
//ACTUALIZACIÓN DE SENSORES
```

```
timer_gps.update();
timer_dht11.update();
timer_mq2.update();
```

- Actualización de los tickers, cada vez que finaliza el tiempo predispuesto para cada función, reproduce la función asociada a cada ticker y vuelve a esperar hasta el siguiente ciclo

```
//GPS
/* OPCION_1
while (gpsSerial.available())
  if (gps.encode(gpsSerial.read()))
    displayInfo();

if (millis() > 5000 && gps.charsProcessed() < 10)
{
  Serial.println(F("No GPS detected: check wiring."));
  while(true);
}
*/
/* OPCION_1.2
while (gpsSerial.available() > 0)
  if (gps.encode(ss.read()))
  {
    displayInfo();
    if (gps.location.isValid())
    {
      latitude = gps.location.lat();
      lat_str = String(latitude , 6);
      longitude = gps.location.lng();
      lng_str = String(longitude , 6);
      Serial.println(lat_str + lng_str);
    }
  }
}
```

```

*/
/* OPCION_2
while (gpsSerial.available() > 0) {
    if (gps.encode(gpsSerial.read())) {
        printData();
    }
}
*/

```

- Funciones para imprimir los datos del gps, de tres formas distintas, comentadas debido a que no se usan.

```

WiFiClient client = server.available();
if (client) //Si hay un cliente presente
{
    Serial.println("Nuevo Cliente");

    //esperamos hasta que hayan datos disponibles
    while(!client.available() && client.connected()){
        delay(1);
    }

    // Leemos la primera línea de la petición del cliente.
    String lineal = client.readStringUntil('r');
    Serial.println(lineal);

    if (lineal.indexOf("CALOR=ON")>0){ //Buscamos un CALOR=ON en la
1ª Línea
        for (int angulo = 0; angulo <= 90; angulo += 1){
            myservo1.write(angulo);
            myservo2.write(angulo);
            delay(10);
        }
    }

    if (lineal.indexOf("FRIO=OFF")>0){ //Buscamos un FRIO=OFF en la
1ª Línea
        for (int angulo = 90; angulo >= 0; angulo -= 1){
            myservo1.write(angulo);
            myservo2.write(angulo);
            delay(10);
        }
    }

    if (lineal.indexOf("BAJARSE")>0){
        for (int angulo = 0; angulo <= 180; angulo += 1){

```



```

        myservo1.write(angulo);
        myservo2.write(180-angulo);
        delay(20);
    }
    delay(3000);
    for (int angulo = 180; angulo >= 0; angulo -= 1){
        myservo1.write(180-angulo);
        myservo2.write(angulo);
        delay(20);
    }
}

if (linea1.indexOf("SUBIRSE")>0){
    for (int angulo = 0; angulo <= 180; angulo += 1){
        myservo1.write(180-angulo);
        myservo2.write(angulo);
        delay(10);
    }
    delay(3000);
    for (int angulo = 180; angulo >= 0; angulo -= 1){
        myservo1.write(angulo);
        myservo2.write(180-angulo);
        delay(10);
    }
}

client.flush();

Serial.println("Enviando respuesta...");
//Encabesado http
client.println("HTTP/1.1 200 OK");
client.println("Content-Type: text/html");
client.println("Connection: close");// La conexión se cierra
después de finalizar de la respuesta
client.println();
//Pagina html para en el navegador
String s="<!DOCTYPE HTML>";
s+="

```

```

s+="  
<br/>";
s+="

```

Esta última parte del loop es la correspondiente a la creación del servidor en el esp8266. El terminal nos proporciona la ip que usa el ESP8266 y lo introducimos dentro de nuestro navegador. Se mostrará en pantalla los valores de temperatura, humedad y nivel de gas que hemos calculado previamente en variables globales.

A parte de mostrar estos valores, podemos hacer que los servomotores de las puertas y ventanas del autobús se muevan. Todo esto se hace mediante unos botones que hemos incluido en la página del servidor, al presionar en los botones dan un valor en nuestro código y dependiendo del botón pulsado podemos hacer que se abran las puertas con los servos si desea bajarse o si hace frío o calor bajar o subir las ventanas.

Toda la página que realiza monitoriza valores y realiza acciones mediante botones lo realizamos mediante HTML.