

Solución IoT basada en HTTP

Parte 2

Introducción

Este proyecto se viene desarrollando desde una guía anterior llamada: [Solución IoT basada en HTTP. Parte 1](#). Es esencial haber desarrollado, esa guía anterior con el fin de poder comprender de qué se trata esta nueva guía. Pero podemos resumirlo así:

En la guía anterior se planteó un reto para desarrollar en varios sprints. El reto es:

Implementar un sistema IoT básico que consiste en dispositivos remotos censando información y enviándola a una base de datos en la nube. Para lo cual se cuenta con un web service propio (hecho por el semillero IoT de la UIS) que permite realizar registros de información en una base de datos basada en Google Sheet, desde cualquier lugar remoto

Con la guía anterior se desarrolló el primer sprint que consistió en:

Una solución completa pero simulando el sensado y usando Python en el llamado remoto

En esta guía, se busca avanzar en un nuevo sprint buscando acercarnos más a cumplir el reto.

Objetivos y competencias a reforzar

- El estudiante reforzará sus conocimientos sobre Arduino y aprenderá a usarlo para consumir un webservice.

Prerrequisitos

El estudiante debe:

- Haber desarrollado con éxito la [Solución IoT basada en HTTP. Parte 1](#) porque será el punto de partida para esta nueva guía.

Conocimientos previos

Métodos Ágiles

Los métodos ágiles son un enfoque de gestión de proyectos que se centra en la entrega incremental de valor, adaptándose rápidamente a los cambios y fomentando la colaboración

continúa entre los miembros del equipo. En lugar de seguir un plan rígido de principio a fin, los proyectos ágiles se desarrollan en ciclos cortos y enfocados llamados "sprints".

Cada sprint tiene una duración fija y un objetivo claro: completar una parte específica del proyecto que pueda ser revisada y mejorada. Este enfoque permite a los equipos responder a las necesidades emergentes, corregir el rumbo si es necesario, y entregar resultados funcionales en menos tiempo.

El uso de sprints es fundamental para dividir un proyecto complejo en tareas manejables, promoviendo un progreso constante y visible. En esta guía, trabajaremos en sprints para avanzar hacia una solución IoT completa, permitiéndote experimentar de primera mano cómo los métodos ágiles facilitan el desarrollo de proyectos efectivos y adaptativos.

Pautas para Dividir un Problema en Sprints

Dividir un problema en sprints es una habilidad clave en los métodos ágiles. Aquí tienes algunas pautas para hacerlo de manera efectiva:

1. **Comprende el Problema Completo:** Antes de dividir el problema, asegúrate de tener una visión clara de lo que quieres lograr. Define el objetivo final del proyecto y cuáles son los resultados esperados.
2. **Identifica Entregables:** Desglosa el problema en entregables pequeños y manejables que puedan completarse en un tiempo corto (1-2 semanas). Cada sprint debería tener un resultado tangible y revisable.
3. **Prioriza las Tareas:** No todas las tareas son igualmente importantes. Prioriza las que son críticas para el éxito del proyecto. Trabaja primero en las tareas que crean la base para las demás.
4. **Define Claramente el Alcance del Sprint:** Cada sprint debe tener un alcance bien definido. Evita agregar nuevas tareas en mitad de un sprint. Si surgen nuevas ideas o cambios, agrégalas a la planificación del siguiente sprint.
5. **Establece Criterios de Éxito:** Define qué significa completar un sprint con éxito. Esto podría incluir la funcionalidad completa de una característica específica, la corrección de un problema crítico, o la entrega de un prototipo funcional.
6. **Revisión y Retroalimentación:** Al final de cada sprint, revisa lo que se ha logrado. Recoge retroalimentación y útila para mejorar el enfoque en los próximos sprints.
7. **Mantén la flexibilidad:** Aunque el objetivo es trabajar en sprints definidos, mantén la flexibilidad para adaptarte a cambios inesperados. Los métodos ágiles valoran la capacidad de responder a cambios sobre seguir un plan rígido.

Ejemplo de Aplicación de Sprints

Contexto: Imaginemos que estás desarrollando un sistema IoT para monitorear la calidad del aire en diferentes ubicaciones. El sistema consistirá en sensores distribuidos que envían datos a una base de datos en la nube, donde se pueden analizar y visualizar en tiempo real.

Sprint 1: Configuración de los Sensores y Envío de Datos Simulados

- **Objetivo:** Configurar los sensores para que envíen datos simulados a un servidor.

- **Tareas:**
 - Seleccionar y configurar los sensores.
 - Desarrollar un script que simula la generación de datos.
 - Implementar el envío de datos simulados a un servidor local.
- **Criterios de Éxito:** Datos simulados enviados y almacenados correctamente en un servidor local.

Sprint 2: Integración con la Nube

- **Objetivo:** Enviar los datos de los sensores a una base de datos en la nube.
- **Tareas:**
 - Configurar una base de datos en la nube (por ejemplo, Google Sheets).
 - Modificar el script para que los datos se envíen a la base de datos en la nube.
 - Asegurar que los datos lleguen y se almacenen correctamente.
- **Criterios de Éxito:** Datos enviados y almacenados en la base de datos en la nube, accesibles desde cualquier ubicación.

Sprint 3: Visualización de los Datos en Tiempo Real

- **Objetivo:** Desarrollar un dashboard que muestra los datos de los sensores en tiempo real.
- **Tareas:**
 - Seleccionar una herramienta de visualización (por ejemplo, Google Data Studio).
 - Conectar la base de datos en la nube al dashboard.
 - Configurar gráficos y tablas que muestran los datos en tiempo real.
- **Criterios de Éxito:** Dashboard funcional que muestra los datos de calidad del aire en tiempo real.

Sprint 4: Implementación y Pruebas en el Mundo Real

- **Objetivo:** Implementar el sistema IoT en una ubicación real y realizar pruebas.
- **Tareas:**
 - Instalar los sensores en la ubicación seleccionada.
 - Monitorear la transmisión de datos desde los sensores en tiempo real.
 - Realizar ajustes en la configuración según sea necesario.
- **Criterios de Éxito:** Sistema IoT instalado y operando con éxito en el entorno real.

Arduino

Es una plataforma de prototipado electrónico de código abierto que utiliza hardware y software fácilmente accesibles y programables para crear proyectos interactivos. Es importante aclarar que Arduino no es un lenguaje de programación, es quizá más comparable con gnuradio en el sentido de que usa una IDE donde se desarrolla el software, pero lo hace para unos ciertos tipos de hardware. El parecido está en que Arduino usa un lenguaje muy reconocido que es C/C++ y GNU radio usa también uno conocido que es Python. Pero también hay diferencias porque gnuradio puede conducir a una solución en la que parte corre en un PC y en parte en el hardware, mientras que en el IDE de Arduino, se

prepara el software que luego, mediante una orden, baja al 100% al hardware para que el hardware que programado para operar independientemente de un computador, pero si tiene algún printout lo envía al computador para visualizar (si es que hay un computador conectado).

Arduino IDE

Significa Entorno de desarrollo integrado. Es el software que un programador usa cuando desea crear código, en un lenguaje de alto nivel, como C++, para que desde allí pueda ser compilado y cargado en una placa en forma de un código de más bajo nivel.

ESP32

Es un microcontrolador de bajo costo y bajo consumo de energía, diseñado para aplicaciones de Internet de las cosas (IoT). Fue desarrollado por la compañía china Espressif Systems y es una evolución del popular ESP8266. El ESP32 tiene dos núcleos de procesamiento de 32 bits, una amplia variedad de periféricos integrados, soporte para conectividad inalámbrica Wi-Fi y Bluetooth, y una gran cantidad de memoria flash y SRAM. El ESP32 se ha convertido en una plataforma popular para proyectos de IoT, robótica y automatización, debido a su bajo costo, bajo consumo de energía, capacidad de procesamiento y conectividad inalámbrica integrada. También cuenta con una comunidad activa de desarrolladores que han creado bibliotecas y herramientas para facilitar su uso y programación.

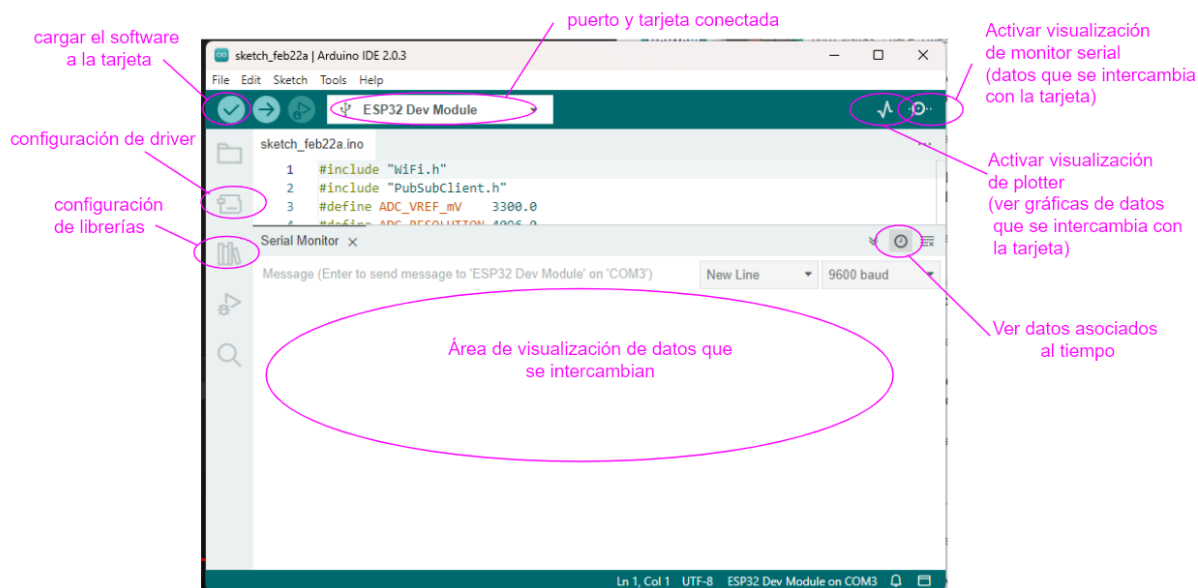
Todos los preparativos cuando no se cuenta con el IDE de Arduino y se trabaja en Windows

Instalación de Arduino en Windows 10 o superior

1. Bajar el instalador así:
arduino.cc/en/software > Windows Win10 and newer > DOWNLOAD
2. Instalación así:
Corres el archivo .exe bajado > Cualquiera que utilice este ordenador > aceptar > aceptar > instalar > aceptar condiciones con condiciones amplias

> seguir aceptando para instalar componentes como Puertos de Adafruit Industries LLC; Arduino USb Drivers; Genuino USb Drivers; dpinst

Interfaz gráfica de la IDE de Arduino



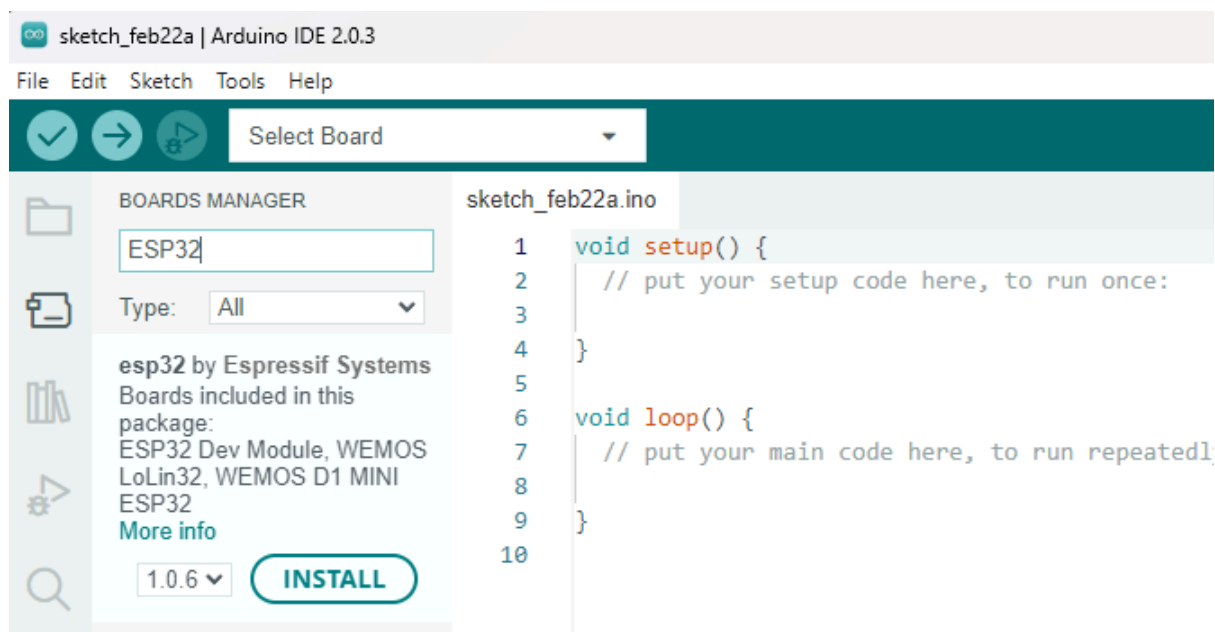
Configuración del IDE Arduino para ESP32

1. Configuración de la URL que contiene el paquete para el ESP-32

abrir el Arduino IDE > Menu horizontal > File > Preferencias > Additional boards Manager URLs: https://dl.espressif.com/dl/package_esp32_index.json > OK

2. Instalación de drivers para la board ESP32

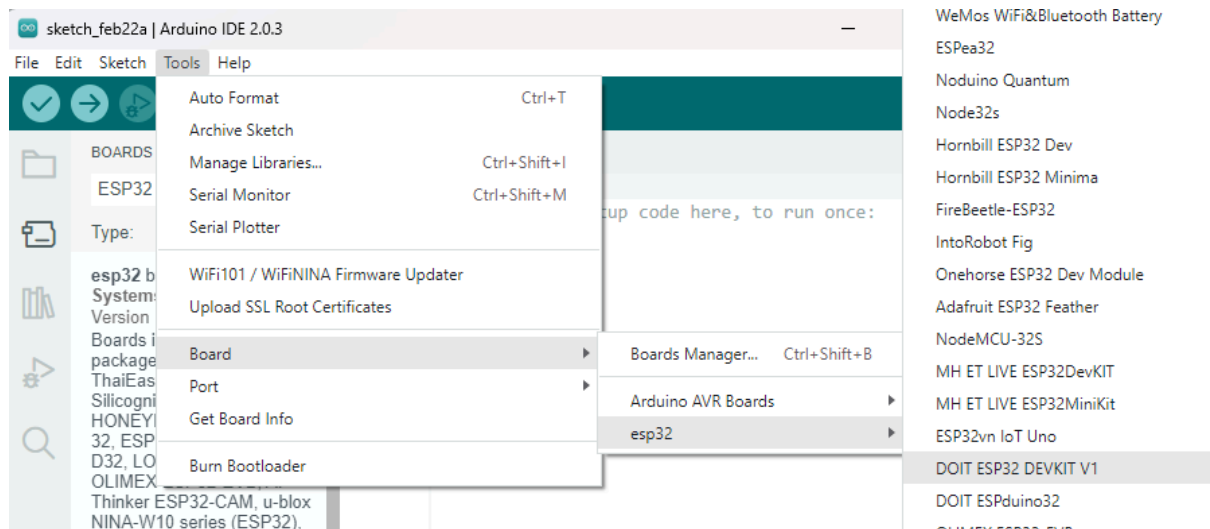
Opcion 1: Barra Menu lateral izquierda >  > en el buscador escribes ESP32 > seleccionas esp32 by Espressif System > Install



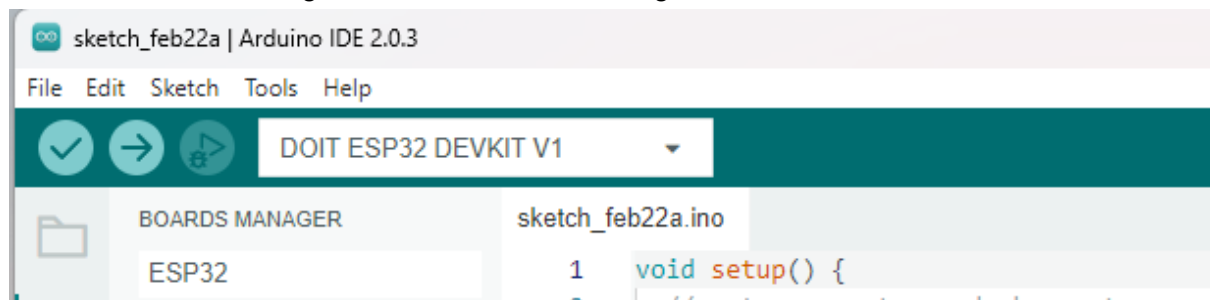
Opcion 2: Tools > Board > Boards Manager > Settings

3. Configuración del Depurador (Debugger) para compilar de C al código de bajo nivel para la ESP32

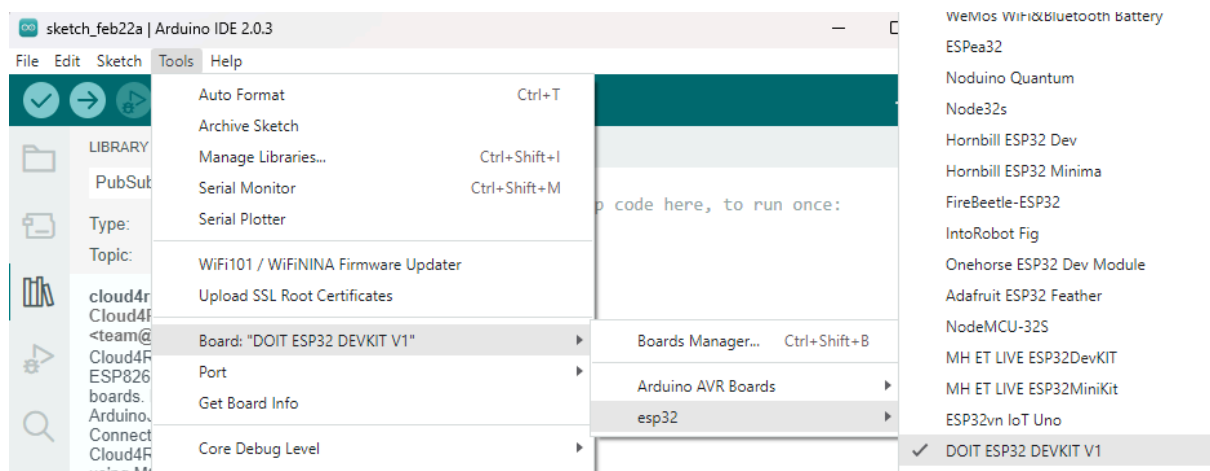
Menu horizontal > Tools > Board > esp 32 > DOIT ESP32 DEVKIT V1



Una vez hecha la configuración tenemos esta imagen en el IDE



Si volvemos a Tools > Board: vemos los datos seleccionados así



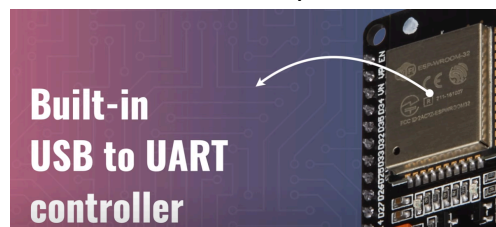
Configuración de Windows para soportar la ESP32 por USB

Para que sea posible instalarle programas a la ESP32 (en inglés “flash programs” en español flashear los programas en la tarjeta) es necesario instalarle drivers. No es necesario agregarle otro hardware porque esa tarjeta viene lista para enchufar por cable USB y listo.

la siguiente es la foto ampliada del controlador donde se aprecian características como la presencia de WiFi

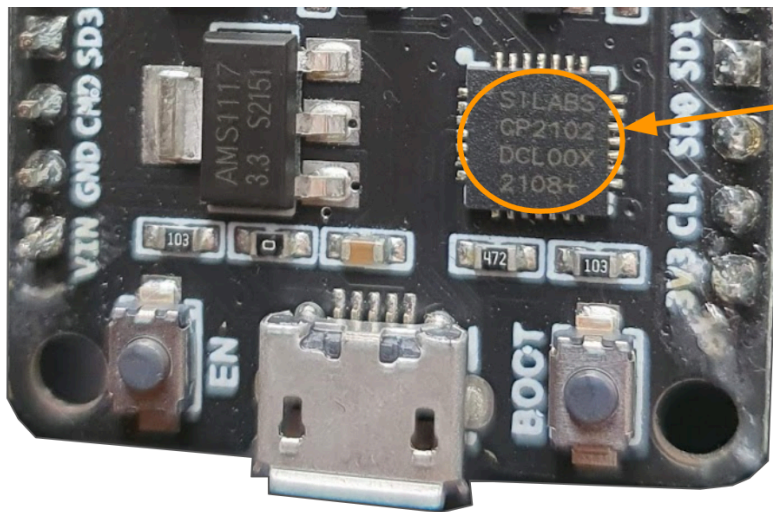


Hay que tener en cuenta que la ESP32 tiene un UART (Universal Asynchronous Receiver/Transmitter). Ese transceiver respeta el protocolo de comunicación que viaja por el puerto USB. ¿Para qué si el dispositivo tiene WiFi?. La comunicación WiFi es la que se usa cuando el dispositivo está siendo usado como un dispositivo IoT, el puerto USB es para facilitar la programación del dispositivo desde un computador. En otras palabras, el dispositivo se puede conectar directamente a un puerto USB de un computador.



Pero falta algo, es como cuando los camiones de servientrega viajan por las mismas carreteras que los demás autos, pero necesitan que en el punto de partida y en el de llegada los reciba la empresa y gente para empacar o desempacar la mercancía, a eso se refiere el controlador USB a UART almacenado en el ESP32, pero se necesita la parte en el otro extremo y eso es lo que usualmente se conoce como el driver.

Es importante identificar el tipo de UART que se tiene, en nuestro caso, contábamos con esta



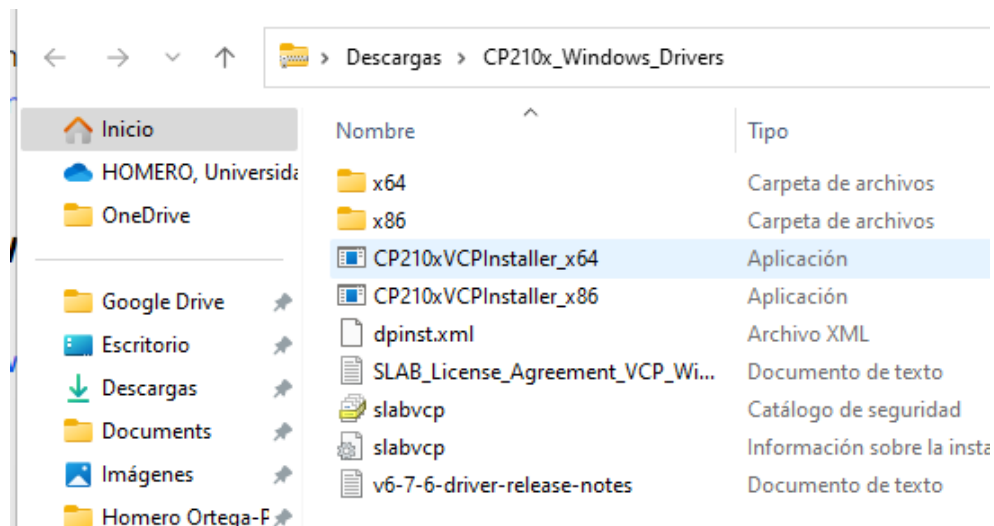
Vemos que la UART es SILABS CP2102 DCLOOX 2108+

- [Video tutorial](#)

En el video vemos que hay 2 tipos de UART y para nuestro caso, el de la figura anterior lo que más se parece es el driver CP210x. En el pie del video se ofrece para este caso el enlace al driver: <https://www.silabs.com/developers/usb-to-uart-bridge-vcp-drivers>

Entonces la secuencia de pasos a seguir son:

<https://www.silabs.com/developers/usb-to-uart-bridge-vcp-drivers> > DOWNLOADS > CP210X Windows Drivers > se instala la opción más apropiada para su computador




Instalación de librerías necesarias para IoT

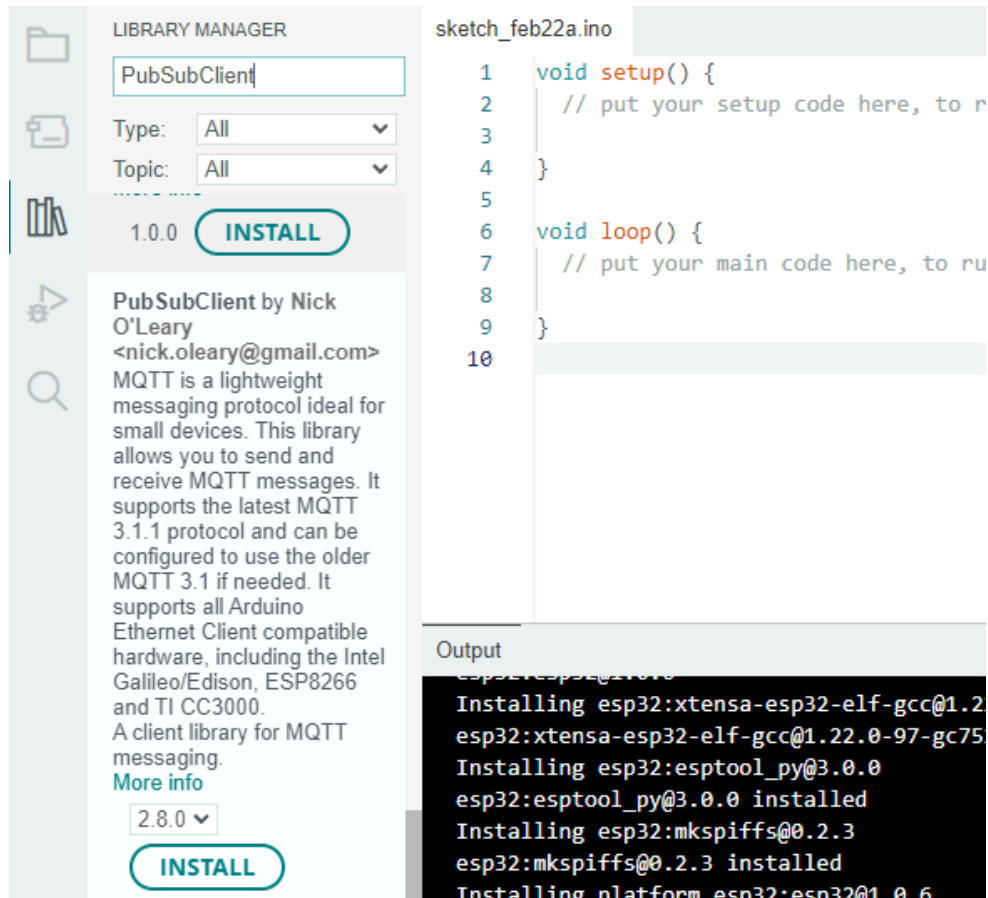
Las dos librerías siguientes son necesarias. La primera para lograr crear una función que permita a la tarjeta quedar conectada a Internet. La segunda es para que crear funciones relacionadas con mqtt:

```
#include "WiFi.h"
```

```
#include "PubSubClient.h"
```


Pasos:

Menú lateral izquierdo >  > en el buscador escribes PubSubCliente > seleccionas PubSubClient by Nick O'Leary > INSTALL



The screenshot shows the Arduino IDE interface. On the left, the Library Manager is open with 'PubSubClient' entered in the search bar. The results show 'PubSubClient by Nick O'Leary' with version 1.0.0 and an 'INSTALL' button. Below the search results, there is a description of the library and a 'More info' link. On the right, the sketch editor shows a simple C++ program with 'void setup()' and 'void loop()' functions. At the bottom, the Output window shows the installation progress for various dependencies like 'esp32:xtensa-esp32-elf-gcc@1.22.0-97-gc75' and 'esp32:esptool_py@3.0.0'.

LIBRARY MANAGER

PubSubClient

Type: All

Topic: All

1.0.0 **INSTALL**

PubSubClient by Nick O'Leary
<nick.oleary@gmail.com>
MQTT is a lightweight messaging protocol ideal for small devices. This library allows you to send and receive MQTT messages. It supports the latest MQTT 3.1.1 protocol and can be configured to use the older MQTT 3.1 if needed. It supports all Arduino Ethernet Client compatible hardware, including the Intel Galileo/Edison, ESP8266 and TI CC3000.
A client library for MQTT messaging.
[More info](#)

2.8.0 **INSTALL**

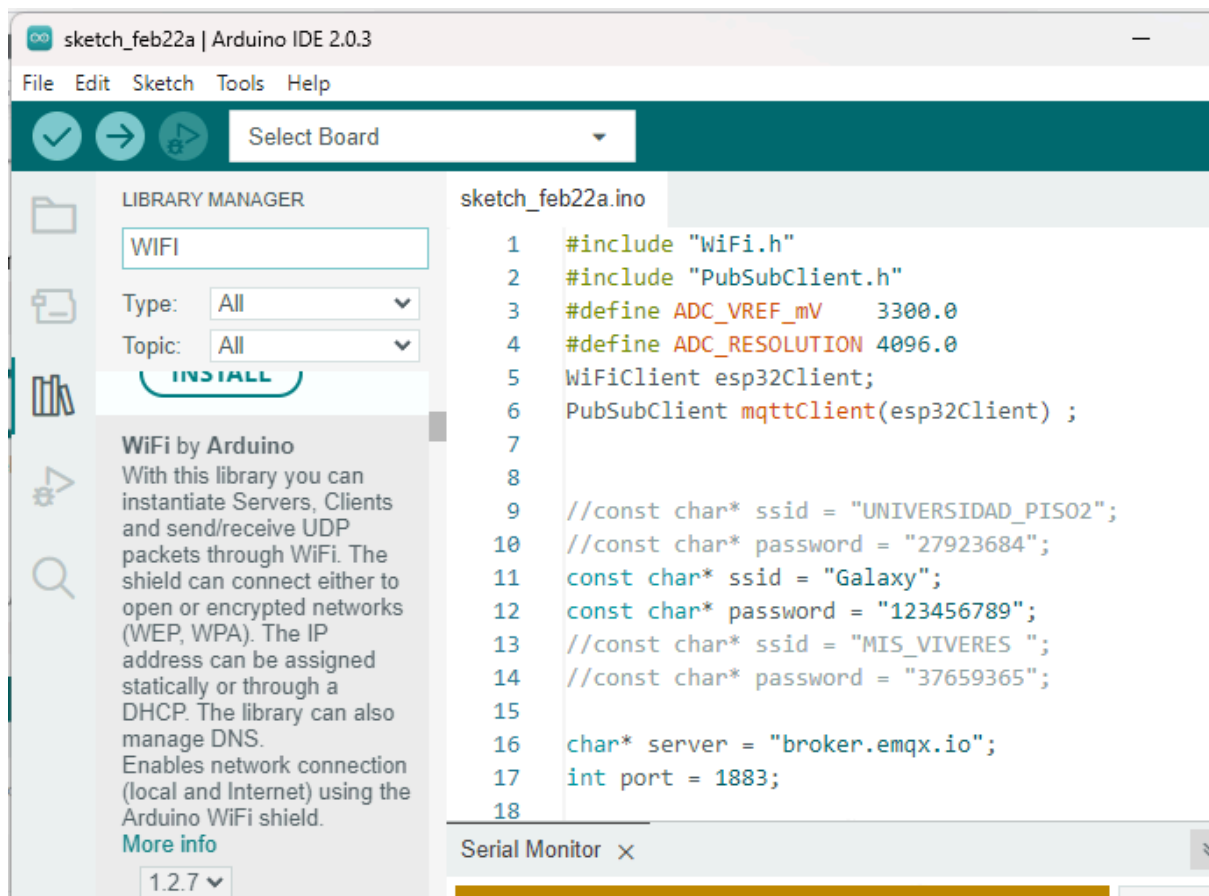
sketch_feb22a.ino

```
1 void setup() {  
2   // put your setup code here, to r  
3  
4 }  
5  
6 void loop() {  
7   // put your main code here, to ru  
8  
9 }  
10
```

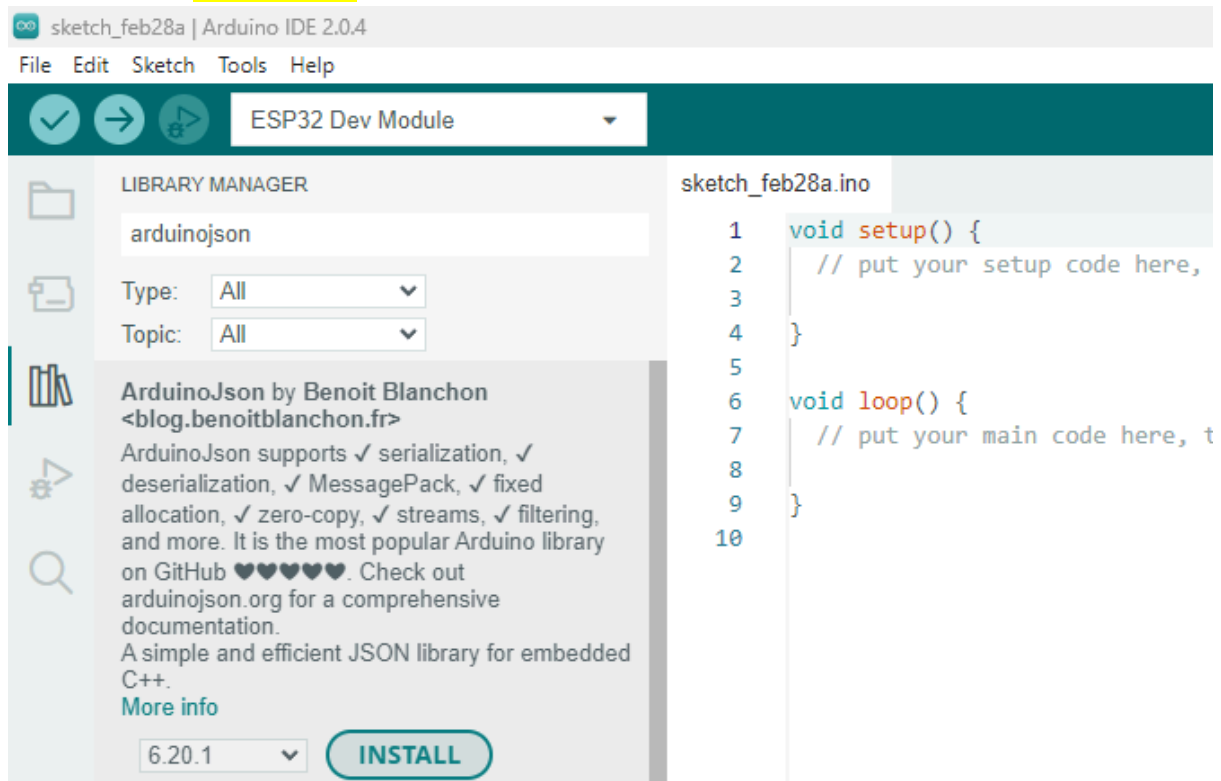
Output

```
Installing esp32:xtensa-esp32-elf-gcc@1.22.0-97-gc75  
Installing esp32:esptool_py@3.0.0  
esp32:esptool_py@3.0.0 installed  
Installing esp32:mkspiffs@0.2.3  
esp32:mkspiffs@0.2.3 installed  
Installing platform_esp32:esp32@1.0.6
```

Hacemos lo mismo para soportar a #Include "WiFi.h"



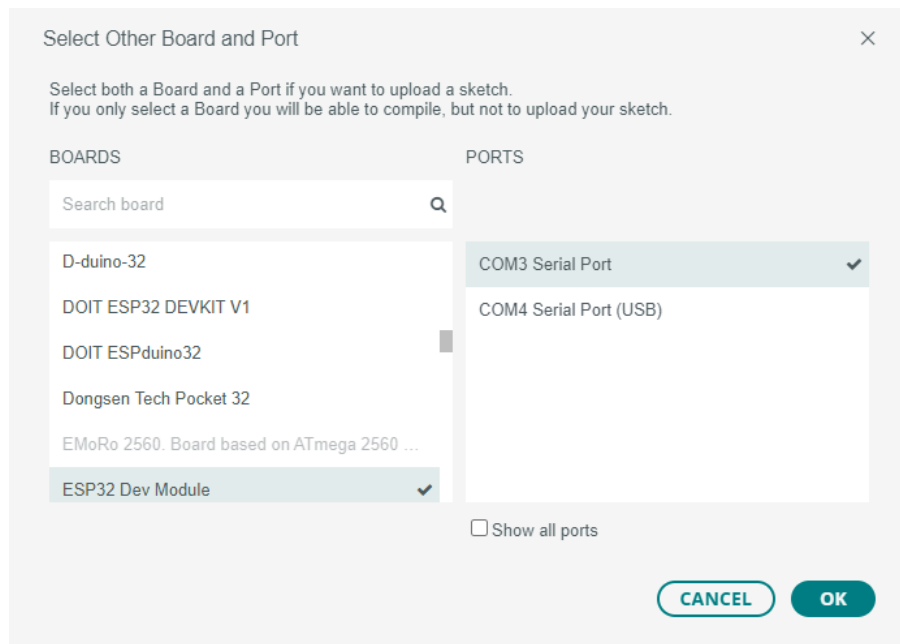
Instalación de **ArduinoJson**



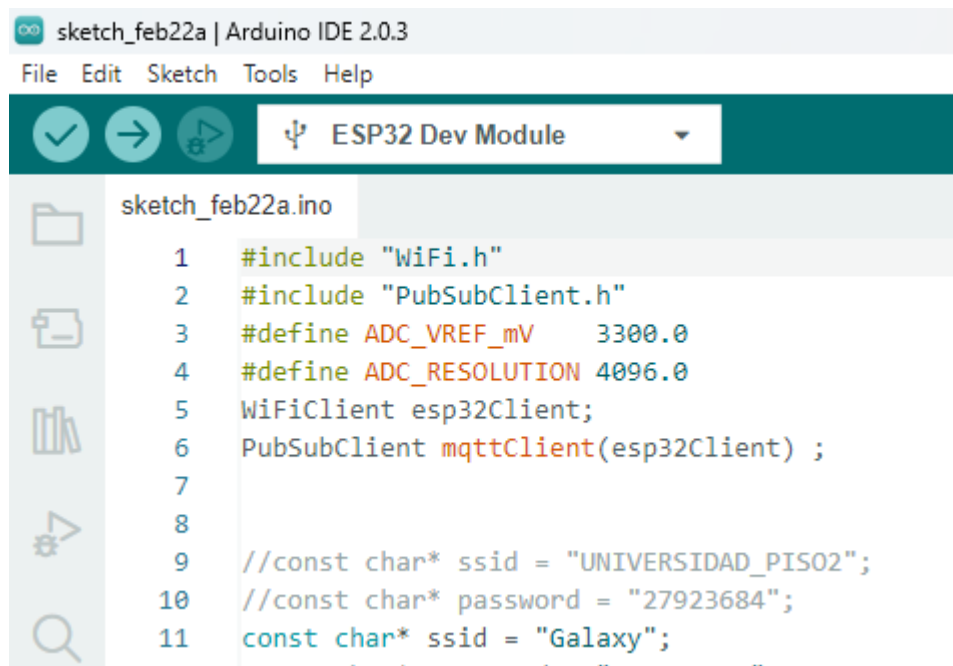
Configuraciones adicionales

Configuración de puertos a usar:

Menu > Select Board > ESP32 Dev module



Como resultado tenemos



Recursos a usar

- Web Service para soportar CRUD sobre Google Sheet: [Documentación del Web Service](#)
- Simulador Wokwi de ESP-32 para personas que no tengan la tarjeta física: https://www.youtube.com/watch?v=K_5u---y-QU

Sprint 2. Solución completa pero simulando el sensado y usando una ESP-32 y Arduino

Paso 1: Conquistar Wokwi con el fin de evitar la instalación del IDE de Arduino en este paso. Comprobar una solución cualquiera usando Woki, siguiendo el [video tutorial](#).

Paso 2: Adaptar la solución anterior para simular los sensores con Arduino en una ESP32 y enviar datos a Google Sheet usando el webservice como se hizo en la guía anterior.

Consejo 1: Al usar Chat gpt el consejo es: pasenle el código hecho en Python, díganle a Chat GPT que ese código funciona maravillosamente en Python. Esto con el fin de que lo use como punto de partida, pero ahora con Arduino y una ESP32.

Consejo 2: observe en el video, que lograr la conexión a WiFi requiere una acción especial, entonces eso hay que enseñárselo a Chat GPT

Consejo 3: No olvide indicar a Chat GPT que igual que se hizo en Python, aquí es necesario simular el sensor.

Código implementado en Arduino

```
#include <WiFi.h>
#include <HTTPClient.h>

// Configuración de la red WiFi
const char* WIFI_NAME = "Wokwi-GUEST";
const char* WIFI_PASSWORD = "";

// URL del web service
const char* webservice_url =
"https://script.google.com/macros/s/AKfycbz7NCKBvRCvhQkuAajcaQUoxhpRQOD_4zOxplrXNP1s2fuiuB-CmPATKADO6ijK__Evm/exec";

// Función simuladorSensor que genera un valor aleatorio
float simuladorSensor() {
    // Genera un valor de temperatura entre 10 y 50 cm
    return random(100, 500) / 10.0;
}

// Función que genera un ID único basado en el timestamp y un valor aleatorio
String generateID() {
    return String(millis()) + String(random(1000, 9999));
}
```

```
void setup() {  
    // Inicializar el monitor serie  
    Serial.begin(115200);  
  
    // Conexión a la red WiFi  
    WiFi.begin(WIFI_NAME, WIFI_PASSWORD);  
    Serial.print("Conectando a WiFi");  
    while (WiFi.status() != WL_CONNECTED) {  
        delay(500);  
        Serial.print(".");  
    }  
    Serial.println(" Conectado!");  
}  
  
void loop() {  
    // Comprobar si estamos conectados a WiFi  
    if (WiFi.status() == WL_CONNECTED) {  
        // Generar la fecha actual y el valor de temperatura  
        String fecha = String(millis());  
        float temperatura = simuladorSensor();  
  
        // Generar un ID único  
        String ID = generateID();  
  
        // Organizar el objeto de datos en formato JSON  
        String datos = String("{\"ordentipo\":\"crear\",\"url\":" +  
String("https://docs.google.com/spreadsheets/d/1xStIjQV--TuZWE0jL5cMnK9HneVwm5BC42n7lgLP4Lc/edit?usp=sharing") +  
String("\",\"numeroHoja\":0,\"filaencabezados\":1,\"columnaId\":2,\"datos\":\"[\\\\" +  
                fecha + String("\\\",\\\") + ID +  
String("\\\",\\\"Sensor_Distancia\\\",\\\"UbicaciónX\\\",\\\"Distancia\\\",\\\"Centímetros\\\",") +  
                String(temperatura) + String("\"]\\\)");  
  
        // Enviar datos al web service  
        HTTPClient http;  
        http.begin(webService_url);  
        http.addHeader("Content-Type", "application/json");  
  
        int httpResponseCode = http.POST(datos);  
  
        // Imprimir el código de respuesta HTTP  
        Serial.print("HTTP Response code: ");  
        Serial.println(httpResponseCode);  
  
        // Imprimir la respuesta del servidor  
        String response = http.getString();  
        Serial.println(response);  
  
        // Enviar un mensaje indicando que los datos fueron enviados  
        Serial.println("Datos enviados al web service.");  
  
        // Cerrar la conexión  
        http.end();
```

```

} else {
    Serial.println("Error en la conexión WiFi");
}

// Esperar 5 segundos antes de la siguiente lectura
delay(5000);
}

```

Resultado

The screenshot shows the Arduino IDE interface. On the left, the sketch is displayed with the following code:

```

1 // Código adaptado tomando como base el código de Daniel Rojas
2
3 #include <WiFi.h>
4 #include <HttpClient.h>
5
6 // Configuración de la red WiFi
7 const char* WIFI_NAME = "Wokwi-GUEST";
8 const char* WIFI_PASSWORD = "";
9
10 // URL del web service
11 const char* webservice_url = "https://script.google.com/macros/s/AKfycbz7NCKBv...";
12
13 // Función simuladorSensor que genera un valor de temperatura aleatorio
14 float simuladorSensor() {
15     // Genera un valor de temperatura entre 18 y 50 cm
16     return random(180, 500) / 10.0;
17 }
18
19 // Función que genera un ID único basado en el timestamp y un valor aleatorio
20 String generateID() {
21     return String(millis()) + String(random(1000, 9999));
22 }
23
24 void setup() {
25     // Inicializar el monitor serie
26     Serial.begin(115200);
27
28     // Conexión a la red WiFi
29     WiFi.begin(WIFI_NAME, WIFI_PASSWORD);

```

On the right, the simulation output is shown:

```

Datos enviados al web service.
HTTP Response code: 302
<HTML>
<HEAD>
<TITLE>Moved Temporarily</TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF" TEXT="#000000">
<!-- GSE Default Error -->
<H1>Moved Temporarily</H1>
The document has moved <A
HREF="https://script.googleusercontent.com/macros/echo?
user_content_key=euTJ5qk7usMsqG6JeGITOwrCvPZY0eTfbKH8_fbHiTP20CgtNjk7MUA
vskwdSV90MPzA7DBPuEl0g8BuDuuU5YIe-
XkQR0Hm5_BxD1H2jW0nuo2oDemN9CCS2h10ox_1xSncGQajx_ryfhECjZEnJbZ0Pq4s7v6z0X
Bxo1kozhw1Cb0xm10XYcS-IWvvnvOrixYxtUn0KO-zvGx5Ih-
27P3llTwzGrhZJ38gYuhH505zSdKwvsQIQ&lib=MreW_OEGtMCGpqd2J3u22NBfbyQXY
-2bv">here</A>.

```

Practica_5_v2

Archivo Editar Ver Insertar Formato Datos Herramientas Extensiones Ayuda

100% 123 Predet... 10 B I A

A43:G45 8355

	A	B	C	D	E	F	G
1	Fecha	ID	Nombre_Sensor	Ubicación	Tipo	Unidades	Valor
34	2024-09-04 05:01:30	20240904050130030662	Sensor_Humedad	Ubicación 2	Humedad	Porcentaje	89.46
35	2024-09-04 05:01:30	20240904050130030680	Sensor_Presion	Ubicación 3	Presión	Pascal	45.62
36	2024-09-04 05:01:43	20240904050143735870	Sensor_Temperatura	Ubicación 1	Temperatura	Celsius	47.77
37	2024-09-04 05:01:43	20240904050143735977	Sensor_Humedad	Ubicación 2	Humedad	Porcentaje	48.43
38	2024-09-04 05:01:43	20240904050143735994	Sensor_Presion	Ubicación 3	Presión	Pascal	71.99
39	2024-09-05 14:45:42	20240905144542954732	Sensor_Temperatura	Ubicación 1	Temperatura	Celsius	93.08
40	2024-09-05 14:45:42	20240905144542954802	Sensor_Humedad	Ubicación 2	Humedad	Porcentaje	56.71
41	2024-09-05 14:45:42	20240905144542954816	Sensor_Presion	Ubicación 3	Presión	Pascal	11.73
42	2024-09-05 14:45:52	20240905144552614743	Sensor_Temperatura	Ubicación 1	Temperatura	Celsius	73.6
43	8355	83557649	Sensor_Distancia	Ubicación X	Distancia	Centímetros	39.5
44	22607	226073313	Sensor_Distancia	Ubicación X	Distancia	Centímetros	15.9
45	36120	361204694	Sensor_Distancia	Ubicación X	Distancia	Centímetros	31.3
46							
47							
48							
49							
50							
51							
52							

+ Hoja 1 CosumoAPI

Paso 3: Mejorar la solución anterior, para incorporar en Wokwi sensores.

Código implementado en Arduino

```
#include <WiFi.h>
#include <HTTPClient.h>
#include <DHT.h> // Librería correcta para el sensor DHT en Wokwi

// Configuración de la red WiFi
const char* WIFI_NAME = "Wokwi-GUEST";
const char* WIFI_PASSWORD = "";

// URL del web service
const char* webservice_url =
"https://script.google.com/macros/s/AKfycbz7NCKBvRCvhQkuAjcaQUoxhpRQOD_4zOxplrXNPls2fuiuB-CmPATKADO6ijK__Evm/exec";

// Pin donde está conectado el DHT22
#define DHTPIN 15
#define DHTTYPE DHT22
DHT dht(DHTPIN, DHTTYPE); // Inicializamos el sensor DHT22

// Función que genera un ID único basado en el timestamp y un valor aleatorio
String generateID() {
    return String(millis()) + String(random(1000, 9999));
}

void setup() {
    // Inicializar el monitor serie
    Serial.begin(115200);

    // Inicializar el sensor DHT22
    dht.begin();

    // Conexión a la red WiFi
    WiFi.begin(WIFI_NAME, WIFI_PASSWORD);
    Serial.print("Conectando a WiFi");
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }
    Serial.println(" Conectado!");
}

void loop() {
    // Comprobar si estamos conectados a WiFi
    if (WiFi.status() == WL_CONNECTED) {
        // Leer los datos de temperatura y humedad del sensor
        float temperatura = dht.readTemperature();
        float humedad = dht.readHumidity();

        // Verificar si la lectura del sensor es válida
        if (isnan(temperatura) || isnan(humedad)) {
            Serial.println("Error al leer del sensor DHT!");
            return;
        }

        // Generar un ID único
    }
}
```

```

String ID = generateID();

// Generar la fecha actual (timestamp)
String fecha = String(millis());

// Organizar el objeto de datos en formato JSON para temperatura
String datos_temp = String("{\"ordentipo\":\"crear\",\"url\":\"")
+
String("https://docs.google.com/spreadsheets/d/1xStIjQV--TuZWE0jL5cMn
K9HneVwm5BC42n71gLP4Lc/edit?usp=sharing") +

String("\",\"numeroHoja\":0,\"filaencabezados\":1,\"columnaId\":2,\"d
atos\":\"[\\\"\\\"") +
        fecha + String("\\\",\\\"") + ID +
String("\\\",\\\"Sensor_DHT22\\\",\\\"pc_Tunja\\\",\\\"Temperatura\\\",
\\\"Celsius\\\",") +
        String(temperatura) + String("]\"}");

// Organizar el objeto de datos en formato JSON para humedad
String datos_hum = String("{\"ordentipo\":\"crear\",\"url\":\"")
+
String("https://docs.google.com/spreadsheets/d/1xStIjQV--TuZWE0jL5cMn
K9HneVwm5BC42n71gLP4Lc/edit?usp=sharing") +

String("\",\"numeroHoja\":0,\"filaencabezados\":1,\"columnaId\":2,\"d
atos\":\"[\\\"\\\"") +
        fecha + String("\\\",\\\"") + ID +
String("\\\",\\\"Sensor_DHT22\\\",\\\"pc_Tunja\\\",\\\"Humedad\\\",\\
\"Porcentaje\\\",") +
        String(humedad) + String("]\"}");

// Enviar datos de temperatura al web service
HTTPClient http_temp;
http_temp.begin(webService_url);
http_temp.addHeader("Content-Type", "application/json");

int httpResponseCode_temp = http_temp.POST(datos_temp);

// Imprimir el código de respuesta HTTP para temperatura
Serial.print("HTTP Response code (Temperatura): ");
Serial.println(httpResponseCode_temp);

// Imprimir la respuesta del servidor para temperatura
String response_temp = http_temp.getString();
Serial.println(response_temp);

// Enviar un mensaje indicando que los datos de temperatura
fueron enviados
Serial.println("Datos de Temperatura enviados al web service.");

// Cerrar la conexión HTTP para temperatura
http_temp.end();

// Enviar datos de humedad al web service
HTTPClient http_hum;

```



```

http_hum.begin(webService_url);
http_hum.addHeader("Content-Type", "application/json");

int httpResponseCode_hum = http_hum.POST(datos_hum);

// Imprimir el código de respuesta HTTP para humedad
Serial.print("HTTP Response code (Humedad): ");
Serial.println(httpResponseCode_hum);

// Imprimir la respuesta del servidor para humedad
String response_hum = http_hum.getString();
Serial.println(response_hum);

// Enviar un mensaje indicando que los datos de humedad fueron
enviados
Serial.println("Datos de Humedad enviados al web service.");

// Cerrar la conexión HTTP para humedad
http_hum.end();
} else {
    Serial.println("Error en la conexión WiFi");
}

// Esperar 5 segundos antes de la siguiente lectura
delay(5000);
}

```

Resultado

The screenshot shows the Wokwi web interface for a project named "prueba_5_SENSOR - Wokwi ES". The code in the left pane is for an ESP32-DHT22 sensor module. It includes functions to send temperature and humidity data to a Google Sheets web service. The right pane shows the simulation output, which includes an HTTP 200 OK response and a confirmation message: "Datos de Humedad enviados al web service."

Se usó como sensor el DHT22, que sensa temperatura y humedad.

Practica_5_v2

Archivo Editar Ver Insertar Formato Datos Herramientas Extensiones Ayuda

100% 123 Predet... - 10 + B I A

K48

	A	B	C	D	E	F	G	H	I	J
1	Fecha	ID	Nombre_Sensor	Ubicación	Tipo	Unidades	Valor			
43	8355	83557649	Sensor_Distancia	Ubicación X	Distancia	Centímetros	39.5			
44	22607	226073313	Sensor_Distancia	Ubicación X	Distancia	Centímetros	15.9			
45	36120	361204694	Sensor_Distancia	Ubicación X	Distancia	Centímetros	31.3			
46	8879	88797796	Sensor_DHT22	Ubicación X	Temperatura	Celsius	24	Humedad	Porcentaje	40
47	25239	252395244	Sensor_DHT22	Ubicación X	Temperatura	Celsius	24	Humedad	Porcentaje	40
48	7883	78827589	Sensor_DHT22	pc_Tunja	Humedad	Porcentaje	40			
49	21950	219495251	Sensor_DHT22	pc_Tunja	Humedad	Porcentaje	40			
50	34962	349617353	Sensor_DHT22	pc_Tunja	Humedad	Porcentaje	40			
51	48473	484728043	Sensor_DHT22	pc_Tunja	Humedad	Porcentaje	40			
52	7885	78857517	Sensor_DHT22	pc_Tunja	Temperatura	Celsius	24			
53	7885	78857517	Sensor_DHT22	pc_Tunja	Humedad	Porcentaje	40			
54	30858	308585321	Sensor_DHT22	pc_Tunja	Temperatura	Celsius	24			
55	30858	308585321	Sensor_DHT22	pc_Tunja	Humedad	Porcentaje	40			
56	53378	533784040	Sensor_DHT22	pc_Tunja	Temperatura	Celsius	24			
57	53378	533784040	Sensor_DHT22	pc_Tunja	Humedad	Porcentaje	40			
58	75273	752736601	Sensor_DHT22	pc_Tunja	Temperatura	Celsius	24			
59	75273	752736601	Sensor_DHT22	pc_Tunja	Humedad	Porcentaje	40			
60										
61										

+ Hoja 1 CosumoAPI

Se puede observar que lo de color morado corresponde a los datos cargados en la primera parte de este taller, enseguida lo de color verde hace referencia a esta parte 2, pero como se puede observar los datos quedan por fuera de la tabla, en lo de color amarillo se arreglo para que los datos queden dentro de la tabla, pero solo se cargaron los datos de la humedad, hizo falta el dato de la temperatura, por eso en la parte de color azul ya se puede observar que funciona correctamente, cargando en el Sheets los datos de temperatura y humedad de cada medición, se sabe porque se repite el ID cada dos datos.