

ARQUITECTURA Y CONECTIVIDAD

Profesor: Jorge Morales

Alumno

- Fernando Gimenez Coria - FerCbr

Módulo II: Familia de Protocolos IoT - II

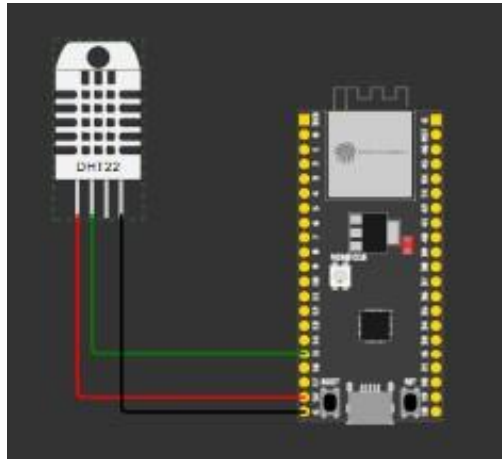
Trabajo Práctico N°4

4) Implementar un código JASON, para comunicar un sensor de temperatura y humedad con un ESP32, Arduino, simulando los mismos en WOKWI,

Proteus, LabView; etc. ¿Cuáles serían los campos mínimos para hacer la implementación?

<https://wokwi.com/projects/429048985375089665>

Para resolver este ejercicio se empleo, en el simulador Wokwi un controlador ESP32 al cual le conectamos un sensor de temperatura y humedad DHT22.



El diagrama de conexiones de detalla a continuación:

```
{
  "version": 1,
  "author": "Anonymous maker",
  "editor": "wokwi",
  "parts": [
    { "type": "board-esp32-s2-devkitm-1", "id": "esp", "top": 0, "left": 0,
"attrs": {} },
    { "type": "wokwi-dht22", "id": "dht1", "top": -28.5, "left": -159,
"attrs": {} }
  ],
  "connections": [
    [ "esp:TX", "$serialMonitor:RX", "", [ ] ],
    [ "esp:RX", "$serialMonitor:TX", "", [ ] ],
    [ "dht1:VCC", "esp:5V", "red", [ "v0" ] ],
    [ "dht1:SDA", "esp:15", "green", [ "v0" ] ],
    [ "dht1:GND", "esp:GND.1", "black", [ "v0" ] ]
  ],
  "dependencies": {}
}
```

El código cargado en el controlador se encarga de:

- ✓ Se conecta a WiFi.
- ✓ Se conecta a un broker MQTT.
- ✓ Lee temperatura y humedad del sensor DHT11.
- ✓ Crea un mensaje JSON con los datos.
- ✓ Publica ese mensaje a un tema MQTT cada 5 segundos.

A continuación analizamos en detalle cada sección:

🔧 1. Inclusión de librerías

```
#include <WiFi.h>                // Para conectarse a una red WiFi
#include <PubSubClient.h>        // Para conectarse a un broker MQTT
#include <DHT.h>                 // Para manejar sensores DHT (DHT11, DHT22, etc.)
#include <ArduinoJson.h>         // Para crear objetos JSON
```

Estas librerías hacen posible:

- La conexión del ESP32 a una red WiFi.
- La comunicación con un broker MQTT.
- La lectura del sensor DHT11.
- La creación y serialización de datos JSON.

2. Definiciones y configuración del sensor

```
#define DHTPIN 15
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);
```

- `DHTPIN` indica que el sensor DHT11 está conectado al **GPIO 15** del ESP32.
- `DHTTYPE` define que se está utilizando un sensor del tipo **DHT11**. □ Luego se crea una instancia llamada `dht`.

📶 3. Datos de conexión WiFi y MQTT

```
const char* ssid = "Wokwi-GUEST"; // Nombre de la red WiFi
const char* password = "";         // Contraseña de la red (está vacía porque Wokwi no la requiere)

const char* mqtt_server = "test.mosquitto.org"; // Dirección del broker MQTT
const int mqtt_port = 1883;                  // Puerto estándar MQTT
const char* mqtt_topic = "iot/esp32/ambiente"; // Tema MQTT al que se enviarán los datos
```

- La red WiFi usada es la que provee **Wokwi**, un simulador online.
- El broker MQTT usado es **test.mosquitto.org**, uno de los más usados para pruebas. □ Los datos se enviarán al tema `"iot/esp32/ambiente"`.

🔌 4. Inicialización de clientes WiFi y MQTT

```
WiFiClient espClient;
PubSubClient client(espClient);
```

- Se crea un cliente WiFi (`espClient`).
- Este cliente es pasado al cliente MQTT (`client`), que usará esa conexión para comunicarse con el broker.

🌐 5. Conexión a la red WiFi

```
void setup_wifi() {
  WiFi.begin(ssid, password); while
(WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("\nWiFi conectado"); }
```

Este bloque conecta el ESP32 a la red WiFi:

- Llama a `WiFi.begin()` con las credenciales.
- Espera hasta que `WiFi.status()` indique que está conectado (`WL_CONNECTED`). □
Imprime puntos mientras espera y un mensaje al conectarse.

🔄 6. Reconexión al broker MQTT (si es necesario)

```
void reconnect() {
  while (!client.connected()) {
    Serial.print("Conectando a MQTT...");
    if (client.connect("ESP32Client")) {
      Serial.println("conectado");
    } else {
      Serial.print("falló, rc=");
      Serial.print(client.state());
      Serial.println(" intentando en 5s...");
    }
    delay(5000);
  }
}
```

Esta función:

- Intenta conectar el ESP32 al broker MQTT.
- Si falla, imprime el código de error y reintenta cada 5 segundos. □ El identificador del cliente MQTT es `"ESP32Client"`.

⚙️ 7. Función de configuración (`setup()`)

```
void setup() {
  Serial.begin(115200);
  dht.begin();    setup_wifi();
  client.setServer(mqtt_server, mqtt_port); }
```

- Inicia la comunicación serie (para depuración).
- Inicia el sensor DHT11.
- Conecta a WiFi.
- Configura el servidor MQTT.

🔄 8. Función principal de ejecución (loop())

```
void loop()
{
  if (!client.connected()) {
    reconnect();
  }
  client.loop();
}
```

- Revisa si el ESP32 sigue conectado al broker. Si no lo está, intenta reconectar.
- `client.loop()` mantiene viva la conexión MQTT y procesa mensajes entrantes (aunque este sketch **solo publica**, no suscribe).

🔧 9. Lectura de sensores y creación del JSON

```
float temp =
dht.readTemperature();    float hum =
dht.readHumidity();

if (isnan(temp) || isnan(hum)) {
  Serial.println("Error al leer el DHT11");
  return; }
}
```

- Lee temperatura y humedad del DHT11.
- Si la lectura falla (devuelve NaN), muestra error y sale del `loop()`.

📦 10. Empaquetado de datos en JSON

```
StaticJsonDocument<200> doc;
doc["sensor"] = "DHT11";    doc["device_id"]
= "ESP32_001";
doc["timestamp"] = millis();    // Tiempo en milisegundos desde que
arrancó el programa    doc["temperature"] = temp;    doc["humidity"] =
hum;
```

```
char buffer[256];    size_t n =
serializeJson(doc, buffer);
```

Se arma un objeto JSON como este:

```
{
  "sensor": "DHT11",
  "device_id": "ESP32_001",
  "timestamp": 123456,
  "temperature": 24.5,
  "humidity": 55.3
}
```

- El JSON se guarda en un buffer llamado `buffer` para luego enviarse por MQTT.

11. Envío del JSON por MQTT

```
Serial.println("Enviando JSON:");
Serial.println(buffer);

client.publish(mqtt_topic, buffer);
```

- Muestra el JSON por el monitor serie.
- Publica el mensaje en el tema `iot/esp32/ambiente`.

zzz 12. Espera entre envíos

```
delay(5000); // cada 5 segundos
```

- Espera 5 segundos antes de volver a ejecutar el `loop()`.

En cuanto a la página para poder visualizar los datos adquiridos y enviados por MQTT se alojo en Github pages y se sirve en la siguiente dirección.

https://programador-fullstack-iot.github.io/Arquitectura_y_conectividad_grupo_N1/

La interfaz HTML actual es una **aplicación web responsiva** que permite visualizar en tiempo real los datos de **temperatura y humedad** provenientes del ESP32 a través de un broker MQTT. Está

pensada para ser clara, moderna y funcional, usando **estilos personalizados**, **Bootstrap**, y **Lucide Icons**.

Estructura del Proyecto

1. Encabezado (<head>)

- Incluye **metadatos estándar** y un título.
- Se importa:
 - Bootstrap 5 para estilos rápidos y responsivos.
 - lucide.min.css para íconos modernos.
 - La librería mqtt.min.js para conectarse al broker MQTT vía WebSocket.
- Tiene un favicon en SVG que también actúa como pequeño logo visual en la pestaña.

2. Estilos personalizados (<style>)

- Se definen **variables CSS** (--primary, --danger, etc.) para mantener consistencia en colores.
- Los estilos están organizados por secciones:
 - body, .header, .card, .gauge, .status, etc.
- Uso de **gradientes**, **animaciones suaves**, transiciones y media queries para una **experiencia responsiva** y visualmente atractiva.
- Indicadores visuales de estado (ej. .connected, .connecting, .error) con animación tipo pulse.



Cuerpo (<body>)

✓ **Header (Encabezado Principal)**

- Muestra el nombre del sistema: **"Monitor DHT11 - ESP32"**.
- Incluye un logo SVG y un subtítulo explicativo.

✓ **Estado de Conexión MQTT**

- Una sección visual que muestra el estado de la conexión al broker MQTT.

- Usa un punto de color que cambia según el estado:
 -  Amarillo animado: Conectando
 - Verde: Conectado
 -  Rojo: Error

Dashboard de Sensores

- Sección con dos tarjetas (card): 1.

Temperatura

- Ícono de termómetro, valor numérico, barra de progreso tipo gauge.
- La clase del valor cambia de color según el valor (hot, cold).

2. Humedad

- Ícono de gotas, valor numérico, gauge.
- Colores dinámicos (dry, humid).