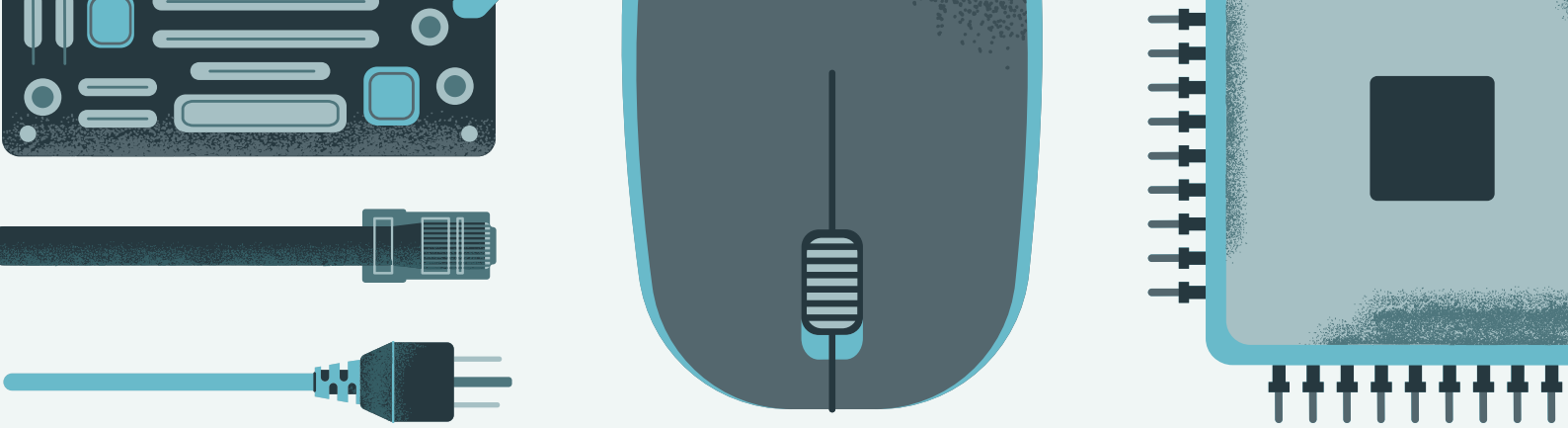

Evidencia de aprendizaje N° 1

TECNICATURA SUPERIOR EN DESARROLLO DE SOFTWARE

APROXIMACIÓN AL MUNDO DEL TRABAJO

Integrantes:

- Battista Mariano Ivan – DNI: 38002217
- Murua Ayosa Christian Jose – DNI: 35674639
- Ayala Jeremias Exequiel – DNI: 42985088

LINK AL PROYECTO:
HTTPS://DOCS.GOOGLE.COM/DOCUMENT/D/1CZ9COIZCNX3TI_CBVRRTNKCVS92CPAZU/EDIT

Desarrollo de Prototipos IoT con ESP32

Proyecto N.º 1: Control de iluminación automático

El presente proyecto tiene como objetivo implementar un sistema de control de iluminación automatizado basado en la cantidad de luz ambiental, utilizando la plataforma **ESP32** y un sensor **LDR (resistor dependiente de la luz)**. Este sistema es útil para optimizar el uso de energía, encendiendo una lámpara o LED solo cuando el ambiente tiene poca luz. Se desarrolló un prototipo funcional en el simulador **Wokwi** y se integró con la plataforma **MQTT** para la publicación de estados de encendido y apagado.

El siguiente enlace corresponde al proyecto en el simulador Woki:
https://wokwi.com/projects/440477318151727105

Y para la visualización de los mensajes en el servidor MQTT se deben seguir los siguientes pasos:

1- Ingresar al enlace https://www.hivemq.com/demos/websocket-client/
2- Click en el botón "connect"
3- Click en "Subscriptions"
4- Click en "Add new topic subscription"
5- En topic escribir: automatizacion-luces-ESP32
6- Click en "subscribe"
7- En el apartado Messages se podrán visualizar las notificaciones el ESP32

Objetivos Específicos

- Detectar el nivel de luz ambiental mediante un sensor LDR.
- Encender automáticamente un LED o relé cuando la luz ambiental es baja.
- Publicar el estado de la lámpara en un servidor MQTT.
- Desarrollar el prototipo en la plataforma Wokwi usando el ESP32.
- Aplicar conocimientos de programación en MicroPython.

Descripción del Hardware utilizado

Componente	Descripción
ESP32	Microcontrolador principal del sistema.
LDR	Sensor de luz, mide la intensidad lumínica.
Relé	Actúa como interruptor para control de cargas con potencia.
LED	Indicador visual del estado del sistema.
Resistencia	Se utiliza para limitar la corriente de entrada al led evitando que se dañe por posibles picos de tensión.

Esquema de Conexión

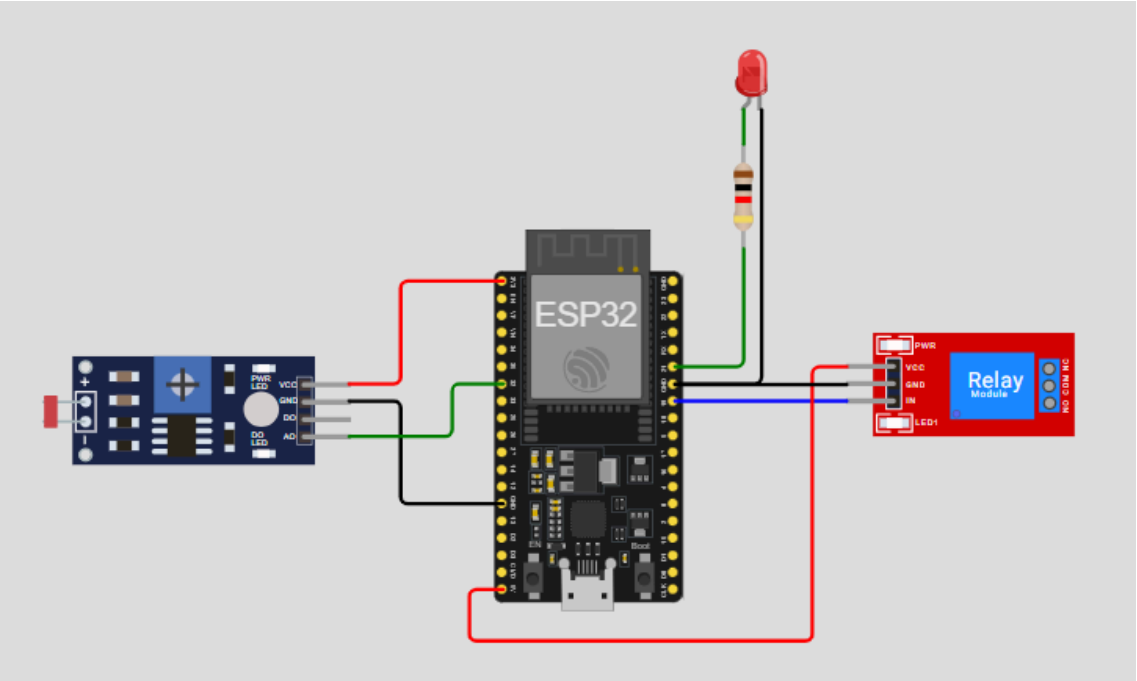

Imagen 1. Captura del simulador Wokwi, de las conexiones entre componentes

Diagrama eléctrico

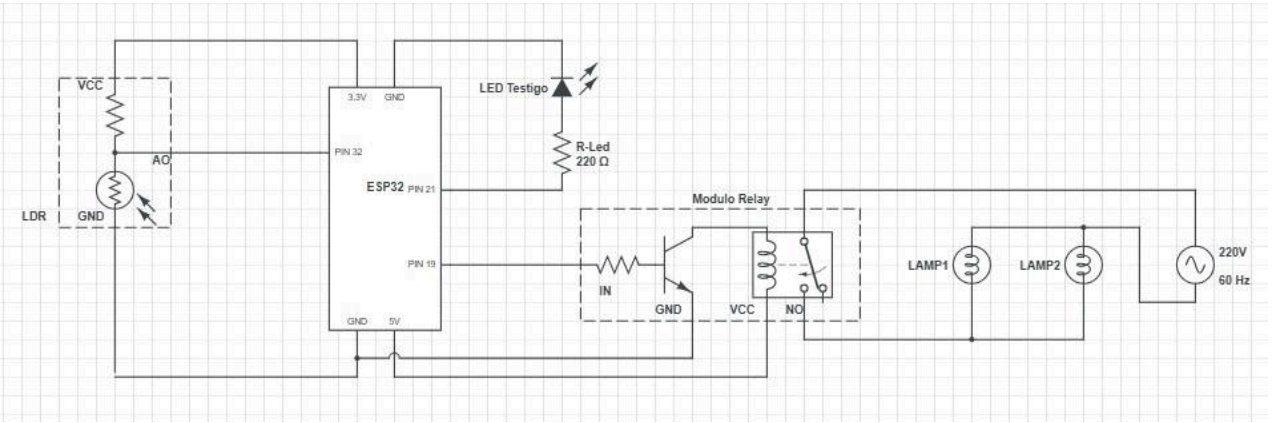

Imagen 2. Captura del diagrama eléctrico del sistema

Conexiones principales

ESP32: Utilizado como microcontrolador central, es el encargado de leer los datos del sensor de iluminación y segun la logica programada permite activar o desactivar un sistema de control de iluminación, además este microcontrolador permite mediante una conexión a red wifi publicar el estado de las luces en un servidor MQTT. Los pines utilizados fueron los siguientes:

GPIO32: entrada analógica para lectura de los datos del LDR.
GPIO19: Configurado como OUT para la señal del modulo relay
GPIO21: Configurado como OUT para encender el LED testigo

LDR (Light Dependent Resistor): es una resistencia cuyo valor varía según la intensidad lumínica. El módulo utilizado integra un divisor resistivo formado por el LDR y un potenciómetro, que genera una señal analógica (proporcional al nivel de luz) disponible en el pin **AO**. Se han conectado los pines del módulo de la siguiente manera:

AO: conectado al **GPIO32** del ESP32
VCC: conectado a 3.3V (del ESP32)
GND: conectado a masa (GND del ESP32)

Relay: El módulo Relay es un dispositivo que se utiliza para manejar cargas de mayor potencia a través de una señal débil como la que entrega el ESP32. Este módulo tiene integrado un transistor de tipo NPN que en conjunto con una resistencia actúa como interruptor para energizar la bobina del relay mediante una señal recibida por el pin **IN**. El relay tiene 3 contactos: **COM** (Comun) **NC** (Normal Cerrado) y **NO** (Normal Abierto) la conexión permite un flujo constante

de corriente desde el pin COM al pin NC hasta que se energiza la bobina, mientras la bobina está energizada, el flujo se desvía desde COM a NO
Para nuestro proyecto hemos conectado los Pines de la siguiente manera:

VCC: al pin 5V del ESP32 (necesaria para estimular la bobina del relay)
GND: a masa (pin GND del ESP32)
IN: conectado a **GPIO19** del ESP32

Para fines prácticos utilizaremos los contactos del relay de la siguiente manera:

COM: Conectado a la Fase de la red domiciliaria
NO: Para alimentar con esta Fase las lámparas que queramos encender mientras la bobina está energizada.

LED: Lo utilizamos como testigo de funcionamiento, este diodo Led estará conectado por su cátodo a **GPIO21** del ESP32 con una resistencia de 220 ohm como limitadora de corriente para evitar que se queme con algún posible pico de tensión, el ánodo del Led va conectado a masa (GND).

Código Fuente en MicroPython

A continuación se muestra el código utilizado, comentado y estructurado para su correcta interpretación.

```
import network
from time import sleep
from machine import Pin, ADC
from umqtt.simple import MQTTClient

# Parámetros MQTT
MQTT_CLIENT_ID = "micropython-luces-proyecto1"
MQTT_BROKER    = "broker.mqttdashboard.com"
MQTT_USER      = ""
MQTT_PASSWORD  = ""
MQTT_TOPIC     = "automatizacion-luces-ESP32"

# Configuración WiFi
SSID = "Wokwi-GUEST"
PASSWORD = ""

# Pines
ldr = ADC(Pin(32))
ldr.atten(ADC.ATTN_11DB)
```

```
rele = Pin(19, Pin.OUT)
led = Pin(21, Pin.OUT)

UMBRAL_LUZ = 3500

# Estado conexión
mqtt_activo = False
client = None

def conectar_wifi():
    wlan = network.WLAN(network.STA_IF)
    wlan.active(True)
    wlan.connect(SSID, PASSWORD)
    for _ in range(20):
        if wlan.isconnected():
            print("Conectado a WiFi:", wlan.ifconfig())
            return True
        sleep(0.5)
    print("No se pudo conectar a WiFi. Continuando sin MQTT.")
    return False

def conectar_servidor():
    global client, mqtt_activo
    try:
        client = MQTTClient(MQTT_CLIENT_ID, MQTT_BROKER,
user=MQTT_USER, password=MQTT_PASSWORD)
        client.connect()
        mqtt_activo = True
        print("Conectado al servidor MQTT")
    except Exception as e:
        print("Error al conectar MQTT:", e)
        mqtt_activo = False

def publicar_estado(estado):
    global mqtt_activo
    if mqtt_activo and client:
        mensaje = "Luz encendida" if estado else "Luz apagada"
        try:
            client.publish(MQTT_TOPIC, mensaje)
            print("MQTT:", mensaje)
        except Exception as e:
            print("Error al publicar MQTT:", e)
            mqtt_activo = False
```

```
# Inicio
conectar_wifi()
conectar_servidor()

estado_anterior = None
valor_ldr_anterior = None

while True:
    valor_luz = ldr.read()
    if valor_luz != valor_ldr_anterior:
        print("LDR:", valor_luz)
        valor_ldr_anterior = valor_luz

    estado_actual = valor_luz >= UMBRAL_LUZ

    if estado_actual != estado_anterior:
        if estado_actual:
            rele.on()
            led.on()
        else:
            rele.off()
            led.off()
        publicar_estado(estado_actual)
        estado_anterior = estado_actual

    sleep(1)
```

Funcionalidad general

El controlador ESP32 recibe un valor analogico de la resistencia LDR el cual representa una cierta cantidad de lúmenes en el ambiente, una vez que estos lúmenes estén por debajo de la intensidad preestablecida, el controlador enviará una señal de salida por uno de sus pines permitiendo con esta señal activar la bobina de un relé para poder alimentar las luces a controlar, a su vez encenderemos un led testigo y emitiremos un mensaje de estado a través de un display indicando si las luces se encuentran encendidas o apagadas.

Explicación del Funcionamiento del Código

1. Conexión WiFi

- El ESP32 se configura para conectarse a la red **Wokwi-GUEST**, que es la red simulada en wokwi.com.
- La función conectar_wifi() intenta establecer conexión hasta 20 veces.
- Si logra conectarse, imprime la IP asignada.
- Si no, el sistema sigue funcionando pero sin publicar en el servidor MQTT.

2. Conexión al servidor MQTT

- Se utiliza la librería umqtt.simple para enviar mensajes.
- La función conectar_servidor() crea un cliente MQTT e intenta conectarse al **broker público** broker.mqttdashboard.com.
- Si se conecta, se habilita el envío de mensajes de estado; si falla, se continúa solo con la lógica local.

3. Lectura del sensor de luz (LDR)

- El **LDR (fotorresistencia)** está conectado al pin analógico 32 del ESP32.
- Se configura con ADC.ATTN_11DB para ampliar el rango de medición (0 a 4095).
- Se lee constantemente con ldr.read(), obteniendo un valor proporcional a la luz ambiente:
    - Valores bajos → mucha luz.
    - Valores altos → poca luz (oscuro).

4. Umbral de comparación

- Se define un **umbral** UMBRAL_LUZ = 3500. (5 lux)
- La lógica es:
    - Si valor_luz >= 3500 → ambiente oscuro → encender luces
    - Si valor_luz < 3500 → ambiente iluminado → apagar luces.

5. Control de salidas

- Dos pines de salida controlan los dispositivos:
    - **Pin 19** → **Relé** (Funciona como interruptor en la conexión de las lámparas de 220V).
    - **Pin 21** → **LED** (testigo visual en la simulación).

- Cuando disminuye la iluminación del ambiente por debajo de 5lux, ambos pines se configuran en estado HIGH (on()), y cuando la iluminación está por encima del umbral definido, los pines se configuran en estado LOW (off()).

6. Publicación del estado

- Cada vez que el estado cambia (encendido ↔ apagado), se envía un mensaje por MQTT, para publicar este mensaje utilizamos un servidor gratuito de MQTT (HiveMQ):
    - "Luz encendida"
    - "Luz apagada"

- Esto permite monitorear el sistema de manera remota en cualquier cliente suscrito al **topic** automatizacion-luces-ESP32.

7. Bucle principal

- El programa funciona en un bucle infinito while True:.
- En cada iteración:

1. Lee el valor del LDR.
2. Compara con el umbral.
3. Enciende o apaga el relé/LED según corresponda.
4. Pública el estado si hubo un cambio.
5. Espera 1 segundo (sleep(1)) antes de repetir.

Lógica de Programación

1. **Entrada**: Sensor LDR → mide luz ambiente.
2. **Proceso**: Comparación del valor medido con un **umbral predefinido**.
3. **Salida**:

    - Activa relé (enciende luces reales).
    - Activa LED testigo.
    - Envía mensaje MQTT de estado.

De esta forma, el **ESP32 actúa como un sistema de control automático de iluminación**, que **toma decisiones basadas en el nivel de luz ambiental**, y **comunica el estado a través de Internet**.

Resultados Obtenidos y Pruebas Realizadas

- Se comprobó que el sistema enciende el LED y el relé automáticamente en condiciones de poca luz.
- El valor del umbral de luz se ajustó a **3500** tras varias pruebas para un mejor comportamiento en el simulador, lo que representa un valor lumínico de 5 lux (equivalente a una leve iluminación justo antes de anochecer)
- La conexión MQTT funcionó correctamente en Wokwi, mostrando los mensajes de encendido y apagado.

Posibles Mejoras Futuras

- Agregar una interfaz web o app móvil para monitoreo en tiempo real.
- Implementar control manual remoto mediante comandos MQTT.
- Guardar datos de uso o estadísticas en una base de datos en la nube.
- Ajuste dinámico del umbral de luz según la hora del día.

Conclusiones

Este proyecto permitió aplicar los conocimientos adquiridos en sensores, programación en MicroPython, redes WiFi, y comunicación MQTT. Se logró construir un sistema funcional y práctico que automatiza el encendido de la luz dependiendo de la luminosidad del entorno, promoviendo así un uso eficiente de la energía. Además, se fortaleció el trabajo en equipo y la capacidad de resolver problemas reales con herramientas de ingeniería.

Proyecto N.º 2: Estación meteorológica

Introducción

Este proyecto tiene como finalidad implementar una **estación meteorológica básica** utilizando un microcontrolador **ESP32**, un sensor de temperatura y humedad **DHT22**, una **pantalla OLED SSD1306**, y una simulación por software de un sensor de presión atmosférica. La estación muestra los valores en pantalla en tiempo real y publica los datos a un servidor **MQTT**.

Debido a que el simulador **Wokwi** no incluye un componente para medir la presión atmosférica (como el BMP180), se ha implementado una **simulación software realista** que genera valores de presión atmosférica con variaciones graduales dentro de un rango válido.

El siguiente enlace corresponde al proyecto en el simulador Woki:
https://wokwi.com/projects/440483972610516993

Y para la visualización de los mensajes en el servidor MQTT se deben seguir los siguientes pasos:

1- Ingresar al enlace https://www.hivemq.com/demos/websocket-client/
2- Click en el botón "connect"
3- Click en "Subscriptions"
4- Click en "Add new topic subscription"
5- En topic escribir: estación-meteorologica-ESP32
6- Click en "subscribe"
7- En el apartado Messages se podrán visualizar las notificaciones el ESP32

Objetivos Específicos

- Medir temperatura y humedad del ambiente usando un sensor DHT22.
- Simular lecturas de presión atmosférica mediante software.
- Mostrar los valores en una pantalla OLED.
- Publicar los datos a un broker MQTT de forma periódica.
- Desarrollar el sistema en el simulador Wokwi.

Descripción del Hardware Utilizado

Componente	Descripción
ESP32	Microcontrolador principal del sistema.
DHT22	Sensor de temperatura y humedad digital.
Pantalla OLED (SSD1306)	Pantalla I2C de 128x64 píxeles para visualización.
Cables y protoboard	Contemplados para realizar las conexiones reales de manera física

Simulación **de sensor BMP180**: Como Wokwi no dispone de un sensor de presión atmosférica, se simularon lecturas válidas en el rango de **980 hPa a 1030 hPa**, con cambios suaves y aleatorios cada un determinado tiempo.

Esquema de Conexión

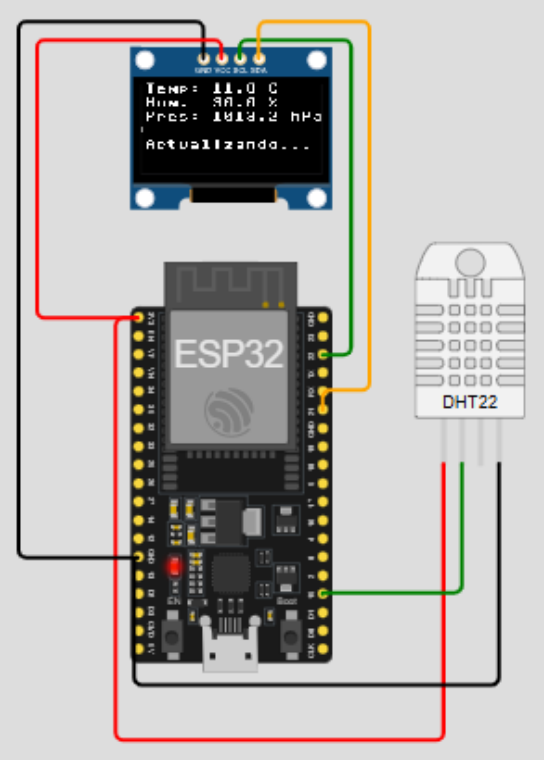

Imagen 3. Captura del simulador Wokwi, de las conexiones entre componentes

Diagrama eléctrico

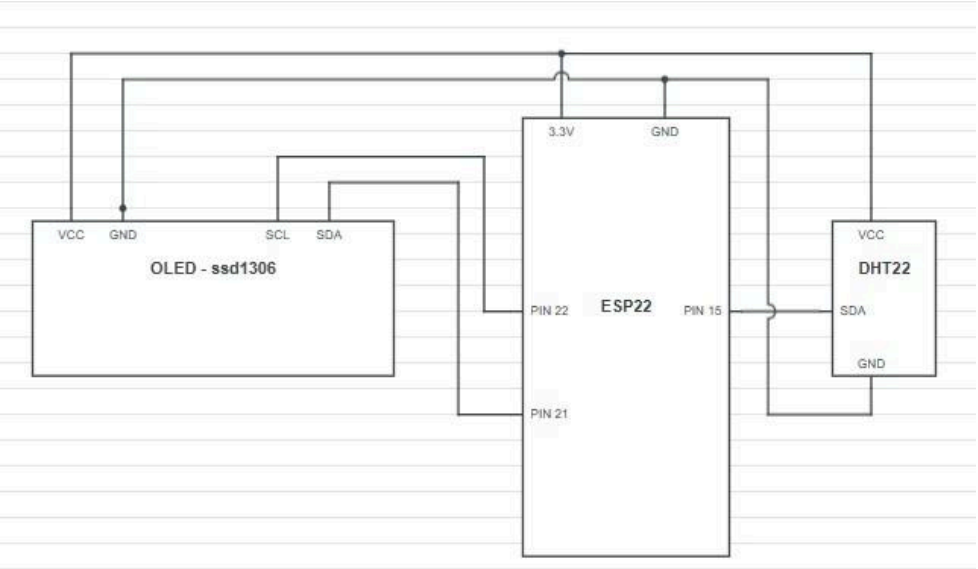

Imagen 4. Captura del diagrama eléctrico del sistema

Conexiones principales

ESP32: Utilizado como microcontrolador central, es el encargado de leer los datos del sensor de temperatura y humedad, procesa estos datos y a través de una comunicación por protocolo I2C los muestra por el display OLED, además este microcontrolador permite mediante una conexión a red wifi publicar estos datos en un servidor MQTT. Los pines utilizados fueron los siguientes:

GPIO15: entrada digital para el DHT22.
GPIO22: salida de reloj I²C (SCL)
GPIO21: entrada/salida de datos I²C (SDA)

DHT22: Es un sensor digital de temperatura y humedad que entrega los datos de forma serial a través de un único pin de señal (**SDA**). Este módulo cuenta con 3 pines principales:

VCC: conectado a 3.3 V del ESP32 para su alimentación.
GND: conectado a masa (GND) del ESP32.
SDA: conectado al **GPIO15** del ESP32, configurado como entrada digital para recibir las lecturas de temperatura y humedad.

El sensor ya incluye la resistencia pull-up necesaria en su línea de datos, por lo que no es necesario agregar componentes adicionales para su funcionamiento.

Pantalla OLED (SSD1306 I2C): La pantalla OLED basada en el controlador SSD1306 utiliza el protocolo de comunicación I2C,el cual permite manejar dispositivos con solo 2 líneas de datos. En este proyecto hemos conectado los pines de la siguiente manera:

VCC: 3.3V del ESP32
GND: a masa (GND del ESP32)
SDA: al GPIO21 del ESP32 configurado como línea de datos I2C
SCL: al GPIO22 del ESP32 configurado como línea de reloj I2C

Código Fuente en MicroPython

A continuación se muestra el código utilizado, comentado y estructurado para su correcta interpretación.

```
import network
import time
from time import sleep
from machine import Pin, SoftI2C
import dht
import urandom
from umqtt.simple import MQTTClient
from ssd1306 import SSD1306_I2C

# Parámetros de conexión MQTT
MQTT_CLIENT_ID = "micropython-luces-proyecto1"
MQTT_BROKER    = "broker.mqttdashboard.com"
MQTT_TOPIC     = "estacion-meteorologica-ESP32"

SSID = "Wokwi-GUEST"
PASSWORD = ""

# Estado de conexión
mqtt_activo = False
client = None

def conectar_wifi():
    wlan = network.WLAN(network.STA_IF)
    wlan.active(True)
    wlan.connect(SSID, PASSWORD)
    for _ in range(20):
        if wlan.isconnected():
```

```
            print("Conectado a WiFi:", wlan.ifconfig())
            return True
        sleep(0.5)
    print("No se pudo conectar a WiFi. Continuando sin MQTT.")
    return False

def conectar_servidor():
    global client, mqtt_activo
    print("Conectando al servidor MQTT... ", end="")
    try:
        client = MQTTClient(MQTT_CLIENT_ID, MQTT_BROKER)
        client.connect()
        mqtt_activo = True
        print("Conectado!")
    except Exception as e:
        print("Error al conectar al servidor MQTT:", e)
        mqtt_activo = False

# Sensor DHT22 en GPIO15
sensor = dht.DHT22(Pin(15))

# Pantalla OLED I2C
i2c = SoftI2C(sda=Pin(21), scl=Pin(22), freq=400000)
oled = SSD1306_I2C(128, 64, i2c)

def show_text(lines):
    oled.fill(0)
    y = 0
    for line in lines[:6]:
        oled.text(line, 0, y)
        y += 10
    oled.show()

# Simulación de presión
PRS_MIN = 980.0
PRS_MAX = 1030.0
prs = 1013.2

def simulate_pressure(prev):
    delta = (urandom.getrandbits(3) - 4) * 0.1
    new = prev + delta
    if new < PRS_MIN:
        new = PRS_MIN + 0.2
```

```
    elif new > PRS_MAX:
        new = PRS_MAX - 0.2
    return new

# Inicialización
show_text(["Iniciando...", "DHT + OLED + BMP180 (Simulado)"])
conectar_wifi()
conectar_servidor()

READ_MS = 2000
PRESSURE_UPDATE_MS = 10000
last_read_ms = 0
last_pressure_ms = 0

ultima_temp = None
ultima_hum = None
ultima_prs = None

while True:
    now = time.ticks_ms()

    if time.ticks_diff(now, last_pressure_ms) >= PRESSURE_UPDATE_MS:
        prs = round(simulate_pressure(prs), 1)
        last_pressure_ms = now

    if time.ticks_diff(now, last_read_ms) >= READ_MS:
        last_read_ms = now
        try:
            sensor.measure()
            t = round(sensor.temperature(), 1)
            h = round(sensor.humidity(), 1)

            if (
                (t != ultima_temp) or
                (h != ultima_hum) or
                (ultima_prs is None or abs(prs - ultima_prs) >= 0.5)
            ):
                show_text([
                    "Temp: {:.1f} C".format(t),
                    "Hum:  {:.1f} %".format(h),
                    "Pres: {:>6.1f} hPa".format(prs),
                    "  ",
                    "Actualizando..."
```

```
            ])

            if mqtt_activo:
                try:
                    mensaje = '{{"temperatura": {:.1f}, "humedad": {:.1f}, "presion":
{:.1f}}}'.format(t, h, prs)
                    client.publish(MQTT_TOPIC, mensaje)
                    print("Publicado en MQTT:", mensaje)
                except Exception as e:
                    print("Error al publicar en MQTT:", e)

            ultima_temp = t
            ultima_hum = h
            ultima_prs = prs

        except OSError as e:
            show_text(["Error", str(e)[:16]])

    time.sleep_ms(10)
```

Explicación del Funcionamiento del Código

1. Conexión a WiFi y servidor MQTT

En la etapa inicial, se simula la conexión a una red Wifi del ESP32 definiendo la red: **Wokwi-GUEST**.
Una vez establecida la conexión, se procede a vincular el dispositivo con un **servidor MQTT público** (broker.mqttdashboard.com).
Esto permite **publicar datos en la nube**, para que puedan ser consultados desde aplicaciones externas.
En caso de que la conexión falle, el sistema continúa funcionando en modo local.

2. Lectura de sensores

El sistema utiliza un **sensor DHT22** conectado al pin GPIO15 para medir dos variables ambientales:

- **Temperatura (°C)**
- **Humedad relativa (%)**

Cada **2 segundos** se realiza una nueva lectura, garantizando una actualización frecuente y precisa de los datos.

3. Simulación de presión atmosférica

Como en la simulación no se dispone de un sensor barométrico real, se implementa una función que genera valores de **presión atmosférica simulada**.
Estos valores se encuentran en el rango de **980 a 1030 hPa**, con pequeñas variaciones aleatorias cada **10 segundos**, lo que imita un comportamiento natural y realista.

Para la aplicación real se utiliza un sensor **BMP180**, este módulo requiere conexión vcc (3.3V), GND y posee una salida analogica (SDA) al igual que el DHT22 el cual se conectaria a un pin del ESP32 configurado como entrada analogica, ademas posee un pin de sincronización tipo I2C que se conectaria junto al PIN 22 al display OLED

4. Visualización en pantalla OLED

Los datos obtenidos se muestran en una pantalla **OLED SSD1306** conectada mediante comunicación I2C (conexión bidireccional que se utiliza para sincronización del ESP32 con los componentes que así lo requieran).
En cada actualización se presentan los valores actuales de temperatura, humedad y presión en un formato legible, además de mensajes de inicio o error en caso de fallos.

5. Publicación de datos en MQTT

Cada vez que se detecta un **cambio significativo** en los valores de temperatura, humedad o presión, se publica un mensaje en formato **JSON** dentro del servidor MQTT en el tópico:

1. estacion-meteorologica-ESP32

Ejemplo del mensaje transmitido:

2. {"temperatura": 25.3, "humedad": 55.1, "presión": 1012.8}

Esto permite que otros dispositivos o plataformas IoT reciban y procesen los datos en tiempo real.

6. Bucle principal

El programa funciona en un bucle infinito, en el cual se ejecutan las siguientes tareas:

1. Leer temperatura y humedad cada 2 segundos.
2. Simular la presión atmosférica cada 10 segundos.
3. Actualizar la pantalla OLED con la información más reciente.
4. Publicar en el servidor MQTT únicamente si se producen cambios relevantes.

Lógica de Programación

1. **Entradas:**
    - Sensor DHT22 → temperatura y humedad.
    - Función simulada → presión atmosférica.
2. **Proceso:**
    - Leer valores periódicamente.
    - Comparar con la última lectura para verificar si hubo cambios significativos.
3. **Salidas:**
    - Mostrar los datos en pantalla OLED.
    - Publicar los datos en formato JSON al broker MQTT (solo si cambian).

Resultados Obtenidos y Pruebas Realizadas

- Las lecturas del DHT22 fueron consistentes en el simulador.
- La simulación de presión atmosférica mostró variaciones graduales dentro de un rango realista.
- El sistema publicó correctamente los datos al servidor MQTT, donde pudieron observarse los valores actualizados.
- La pantalla OLED mostró en tiempo real los parámetros atmosféricos.

Posibles Mejoras Futuras

- Reemplazar la simulación de presión con un sensor real (BMP180/BMP280) en hardware físico.
- Agregar un sensor de luz, lluvia o viento.
- Almacenar los datos en una base de datos (como Firebase o Google Sheets).
- Añadir un botón para forzar la actualización manual de los datos.

Conclusiones

Este proyecto integró varios elementos importantes del desarrollo embebido: sensores físicos, visualización en pantalla, simulación de sensores ausentes y comunicación en red mediante MQTT. A pesar de la limitación del simulador respecto a la medición de presión, se logró una simulación convincente, y se desarrolló un prototipo completo de una estación meteorológica básica.

El trabajo en equipo, la creatividad para superar limitaciones y el uso combinado de hardware y software fueron aspectos clave del desarrollo exitoso de este proyecto.