

Arquitectura y Conectividad

Cuestionario N°3

Profesor Ing. Morales Jorge Elias | <https://github.com/JorEI057>

Miembros:

- Durigutti, Vittorio | GitHub: <https://github.com/vittoriodurigutti>
- Zalazar, Joaquín | GitHub: <https://github.com/breaakerr>
- Marquez, José | Github: <https://github.com/marquezjose>
- Lujan, Luciano | Github: <https://github.com/lucianoilujan>
- Velez, Nahuel | Github: <https://github.com/Lucasmurua19>
- Juncos, Lisandro | Github: <https://github.com/Lisandro-05>
- Garzón, Joaquín | Github: <https://github.com/Joacogarzonn>
- Guzmán, Maria |Github: <https://github.com/lilenguzman01>

Actividad: (Click en las preguntas para link a las respuestas)

1. ¿Cuáles son las peticiones más comunes en el Protocolo HTTP?, ¿Para qué se usan? Ejemplifica.(Indicar por lo menos 5).
2. ¿Cuáles son las principales ventajas de utilizar MQTT en comparación con otros protocolos de comunicación IoT?
3. ¿Cómo se maneja la seguridad en MQTT y cuáles son algunas de las mejores prácticas para garantizar la protección de los datos transmitidos a través de este protocolo?
4. Simular un sensor de temperatura, una luz con dimer, un botón de encendido y apagado mediante Wokwi, Proteus, LabView; etc con ESP32.
5. Implementar un prototipo del dispositivo antes mencionado con ESP32 y conectarlo a un Broker mediante Protocolo MQTT, visualizando en Smartphone o Tablet. En su defecto controlar y comunicar 3 dispositivos, sensores y/o actuadores, mediante el protocolo mencionado. Pueden usar Arduino, pero necesitan el módulo de comunicación a internet. El lenguaje de programación es a su elección, Python, C++, etc.
6. Realizar el Dashboard y producir video de funcionamiento y presentación en *.ppt.
7. Imaginen que tienen una casa inteligente con sensores de temperatura, luces automáticas y una cafetera conectada. ¿Por qué sería conveniente usar el protocolo MQTT para que estos dispositivos se comuniquen entre sí? Mencionen al menos tres características del protocolo que justifiquen su respuesta.

8. Supongamos que un sensor de movimiento instalado en el pasillo publica un mensaje cuando detecta movimiento. Las luces del pasillo están configuradas para encenderse cuando reciben ese mensaje. Explicar cómo funciona esta comunicación usando el modelo publicador/suscriptor de MQTT, e indicar cuál es el papel del broker.

Fecha de presentación: 21/04/25.

1- ¿Cuáles son las peticiones más comunes en el Protocolo HTTP?, ¿Para qué se usan? Ejemplifica.(Indicar por lo menos 5).

El protocolo HTTPS utiliza varios métodos de solicitud (también conocidos como "métodos HTTP") para interactuar con los recursos de un servidor web. A continuación se describen las 5 más comunes, su uso y un ejemplo para cada una:

1. GET

- Uso: Solicita datos de un recurso específico. No debe usarse para enviar datos sensibles.
- Ejemplo: Un usuario accede a una página web.

GET /productos HTTP/1.1
Host: www.ejemplo.com

2. POST

- Uso: Envía datos al servidor, comúnmente para crear nuevos recursos o enviar formularios.
- Ejemplo: Un usuario envía un formulario de contacto.

Host: www.ejemplo.com
Content-Type: application/x-www-form-urlencoded

nombre=Juan&mensaje=Hola

3. PUT

- Uso: Envía datos al servidor para actualizar un recurso existente o crear uno si no existe.
- Ejemplo: Actualizar la información de un producto.

Host: www.ejemplo.com

Content-Type: [application/json](#)

```
{  
  "nombre": "Teclado",  
  "precio": 45.99  
}
```

4. DELETE

- Uso: Elimina un recurso específico del servidor.
- Ejemplo: Eliminar un producto del inventario.

[DELETE /productos/123 HTTP/1.1](#)

Host: www.ejemplo.com

5. PATCH

- Uso: Realiza una actualización parcial a un recurso.
- Ejemplo: Actualizar solo el precio de un producto.

[PATCH /productos/123 HTTP/1.1](#)

Host: www.ejemplo.com

Content-Type: [application/json](#)

```
{  
  "precio": 39.99  
}
```

2- ¿Cuáles son las principales ventajas de utilizar MQTT en comparación con otros protocolos de comunicación IoT?

¿Qué es MQTT?

MQTT (Message Queuing Telemetry Transport) es uno de los protocolos de comunicación más populares en el ámbito del Internet de las Cosas (IoT), y ha ganado reconocimiento por su eficiencia, simplicidad y bajo consumo de recursos. Se trata de un protocolo de mensajería ligero basado en el modelo publicador/suscriptor, que funciona sobre TCP/IP y fue diseñado para conexiones remotas con dispositivos de recursos limitados y redes de baja calidad o alta latencia.

Ventajas frente a otros protocolos:

1. Eficiencia en el uso del ancho de banda

- MQTT utiliza paquetes muy livianos, con encabezados mínimos de solo 2 bytes, lo cual lo hace ideal para redes con poco ancho de banda, como las redes móviles o redes de sensores.
- A diferencia de HTTP, que es más pesado y requiere una conexión continua entre cliente y servidor, MQTT permite ahorrar datos y recursos en cada transmisión.

2. Bajo consumo de energía

- Los dispositivos IoT suelen estar alimentados por baterías, y MQTT está optimizado para minimizar el uso de energía.
- Permite que el dispositivo se conecte, envíe o reciba mensajes y se vuelva a dormir rápidamente, lo cual extiende la vida útil de la batería, en comparación con protocolos como HTTP que requieren mantener la conexión abierta más tiempo.

3. Modelo **publish/subscribe (publicador/suscriptor)**

- Este modelo permite una comunicación desacoplada, es decir, el emisor (publicador) y el receptor (suscriptor) no necesitan conocerse ni mantenerse conectados entre sí directamente.
- Esto reduce la complejidad de los dispositivos y mejora la escalabilidad del sistema. Esto es especialmente útil cuando hay muchos dispositivos conectados.

4. **Mecanismo de calidad de servicio (QoS)**

MQTT permite elegir entre tres niveles de QoS según la importancia del mensaje:

- **QoS 0:** El mensaje se entrega una vez como máximo (sin garantía de entrega).
- **QoS 1:** El mensaje se entrega al menos una vez.
- **QoS 2:** El mensaje se entrega exactamente una vez (máxima garantía).

Esta flexibilidad permite ajustar el comportamiento según las necesidades de cada aplicación IoT.

5. **Persistencia y Retención de mensajes**

- MQTT permite retener mensajes en el broker para que los nuevos suscriptores reciban la información más reciente inmediatamente al conectarse.
- También puede almacenar mensajes offline, lo cual es ideal cuando un dispositivo pierde conexión y necesita recibir los datos que se enviaron mientras estuvo desconectado.

6. **Soporte para conexiones inestables**

- MQTT fue diseñado para entornos con conexiones poco fiables o intermitentes, como redes móviles o rurales.
- El protocolo puede mantener la sesión del cliente, permitiendo la reconexión automática y la recuperación de mensajes perdidos.

7. **Escalabilidad y simplicidad**

- Es fácil de implementar y escalar a miles de dispositivos sin aumentar mucho la carga del sistema.
- A diferencia de otros protocolos como AMQP, que son más robustos pero también más complejos, MQTT ofrece un equilibrio ideal entre simplicidad y funcionalidad para la mayoría de las aplicaciones IoT.

8. **Seguridad (Depende de la implementación)**

- Aunque MQTT por sí solo no incluye cifrado, se puede integrar fácilmente con TLS/SSL para proteger los datos en tránsito.

- Además, permite autenticación mediante usuario y contraseña, lo cual puede ser suficiente en sistemas controlados o cerrados.

9. Compatibilidad con múltiples plataformas y lenguajes

- Hay clientes MQTT para prácticamente todos los lenguajes de programación (Python, C/C++, JavaScript, Java, etc.) y sistemas operativos, incluyendo microcontroladores como ESP32 o Arduino.

Tabla comparativa de diferentes protocolos

Protocolo	Modelo	Peso	QoS	Recomendado para...
MQTT	Pub/Sub	Muy bajo	Sí	Sensores, IoT con bajo consumo.
HTTP	Cliente/Servidor	Alto	No	Web apps, APIs tradicionales.
CoAP	Cliente/Servidor (UDP)	Muy bajo	Sí (limitado)	Redes muy restringidas, sensores pequeños.
AMQP	Pub/Sub	Alto	Sí (avanzado)	Aplicaciones críticas, banca, industria.
WebSocket	Bidireccional	Medio	No	Apps en tiempo real, web/chat.

3- ¿Cómo se maneja la seguridad en MQTT y cuáles son algunas de las mejores prácticas para garantizar la protección de los datos transmitidos a través de este protocolo?

Seguridad en MQTT

La seguridad en MQTT es crucial debido a su uso en sistemas IoT donde se transmiten datos sensibles. Aunque MQTT es un protocolo ligero y eficiente, que no tiene seguridad integrada, por lo que debe complementarse con medidas externas y prácticas:

Manejo de la seguridad en MQTT

1. Autenticación

- Autenticación: Esto garantiza que solo dispositivos autorizados puedan conectarse al broker.
- Implementación: Encriptación, autenticación basada en tokens para mayor seguridad.

Un broker MQTT puede requerir que cada dispositivo IoT se autentique con un nombre de usuario y contraseña antes de publicar o suscribirse.

2. Encriptación

- Datos: Controla qué dispositivos o usuarios tienen permiso para acceder a temas específicos.
- Implementación: Listas de control de acceso (ACLs) que definen permisos por usuario o dispositivo

Configurar el broker para usar `mqtt://` en lugar de `mqtt://`, asegurando la conexión mediante TLS.

3. Autorización

- Datos: Protege los datos durante la transmisión para evitar que sean interceptados.
- Implementación: Uso de TLS (Transport Layer Security) para cifrar la conexión entre cliente y broker, Certificados digitales para verificar la identidad del broker y del cliente

Un LDR IoT puede publicar datos en el tema `casa/LDR`, pero solo dispositivos específicos pueden suscribirse.

4. Prevención de ataques

- Datos: Protege contra amenazas como ataques de denegación de servicio (DoS).
- Implementación: Límites en la tasa de conexiones, Uso de firewalls para restringir acceso no autorizado al broker.

Configurar el broker para limitar la cantidad de mensajes por segundo que un dispositivo puede enviar.

Manejo de la seguridad en MQTT.

Prácticas de seguridad en MQTT:

Utilizar TLS (Transport Layer Security):

- Cifra las comunicaciones entre dispositivos y el broker.
- Asegúrate de usar certificados confiables para evitar ataques de intermediarios.

Autenticación robusta:

- Requiere credenciales únicas para cada dispositivo o usuario.
- Implementa autenticación basada en certificados para mayor seguridad.

Configurar ACLs (Listas de Control de Acceso):

- Define qué dispositivos pueden publicar o suscribirse a temas específicos.
- Evita que dispositivos no autorizados accedan a datos sensibles.

Evitar credenciales en texto plano:

- No almacenes nombres de usuario y contraseñas en texto plano en el dispositivo.
- Utiliza técnicas de hash para proteger credenciales almacenadas.

Limitar permisos y accesos:

- Aplica el principio de privilegio mínimo, permitiendo que cada dispositivo acceda solo a los temas necesarios.
- Por ejemplo, un dispositivo IoT agrícola solo necesita datos del tema campo/sensores.

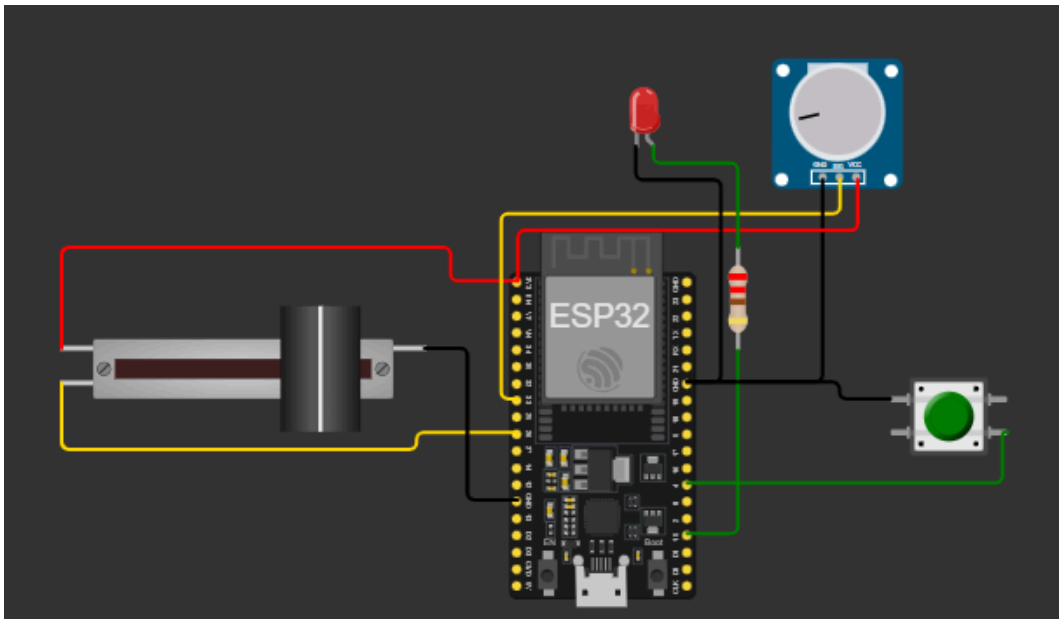
Monitoreo y auditoría:

- Habilita el registro de eventos en el broker MQTT.
- Monitorea intentos fallidos de conexión para detectar posibles amenazas.

Uso de QoS adecuado:

- Ajusta el nivel de calidad de servicio (QoS) según la importancia de los datos.
- Para datos críticos, utiliza QoS 2, que garantiza entrega exacta.

4- Simular un sensor de temperatura, una luz con dimmer, un botón de encendido y apagado mediante Wokwi, Proteus, LabView; etc con ESP32.



El código desarrollado en el simulador Wokwi, y la simulación se puede ejecutar desde la extensión de platformio en VScode. A continuación se describe el funcionamiento parte por parte del código. El prototipo funcional se encuentra alojado en el repositorio correspondiente.

Aquí se explica el desarrollo del mismo:

1. Inclusión de librerías y definición de pines

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

```
// Pines del ESP32
```

```
const int tempSensorPin = 26; // Slide pot como "sensor" de temperatura
```

```
const int dimmerPin = 33; // Potenciometro estándar para regular brillo
```

```
const int ledPin = 15; // LED conectado al pin 15
```

```
const int buttonPin = 4; // Botón conectado al pin 4
```

- **#include <Arduino.h>** da acceso a **analogRead()**, **digitalRead()**, **millis()**, etc.
- Se definen las constantes para cada pin, de modo que si cambias pines en el hardware solo modifiques aquí arriba.

2. Variables de estado

```
int tempRawValue      = 0;    // Lectura bruta ADC del "sensor"

float simulatedTemperatureC = 0.0; // Temperatura en °C tras map()

int dimmerValue       = 0;    // Lectura bruta ADC del dimmer

bool lightOn          = false; // Estado actual del LED (encendido/apagado)

// Variables para debounce del botón

int lastButtonReading = HIGH; // Última lectura cruda del pin

bool lastDebouncedState = HIGH; // Último estado "estable" verificado

unsigned long lastDebounceTime = 0; // Timestamp del último cambio

const unsigned long debounceDelay = 50; // Ms mínimo para considerar el cambio válido

// Para controlar cada cuánto imprimimos en el monitor serie

unsigned long lastPrintTime = 0;

const unsigned long printInterval = 1000; // imprimir una vez por segundo
```

- **Debounce:** evita lecturas falsas (rebotes) del botón.
millis() lo usamos como reloj sin bloquear el programa con **delay()** excesivos.

3. Configuración inicial (**setup()**)

```
void setup() {

    Serial.begin(115200);           // Inicia el puerto serie a 115200 bps

    pinMode(buttonPin, INPUT_PULLUP); // Botón con pull-up interno

    pinMode(ledPin, OUTPUT);        // LED como salida digital
```



```
Serial.println("Sistema listo: Botón, LED con dimmer, sensor simulado.");
```

```
}
```

- **INPUT_PULLUP** activa la resistencia interna que deja el pin en HIGH hasta que el botón lo lleve a GND.
- Inicializamos el monitor serie y damos un mensaje de “arranque”.

4. Bucle principal (**loop()**)

```
tempRawValue = analogRead(tempSensorPin);
```

```
// Mapeamos [0–4095] → [10–40] °C
```

```
simulatedTemperatureC = map(tempRawValue, 0, 4095, 10, 40);
```

- **analogRead()** en el ESP32 devuelve 0...4095 por su ADC de 12 bits.
- **map()** escala ese valor a un rango realista de temperatura.

4.2. Leer y “debouncear” el botón

```
int currentReading = digitalRead(buttonPin);
```

```
if (currentReading != lastButtonReading) {
```

```
    // Si cambió la lectura, reiniciamos el contador de estabilidad
```

```
    lastDebounceTime = millis();
```

```
}
```

```
if ((millis() - lastDebounceTime) > debounceDelay) {
```

```
    // Si el estado se mantuvo estable más de 50 ms:
```

```
    if (currentReading == LOW && lastDebouncedState == HIGH) {
```

```
        // Detectamos un flanco de bajada (pulsación)
```

```
        lightOn = !lightOn; // Toggle del estado de la luz
```

```
}
```

```
lastDebouncedState = currentReading;
```

```
}
```

```
lastButtonReading = currentReading;
```

- **Flanco de bajada:** solo cuando pasamos de HIGH a LOW invertimos `lightOn`.
- Evitamos múltiples toggles por un único clic gracias al retardo de `debounceDelay`.

4.3. Control del LED con PWM (dimmer)

```
if (lightOn) {
```

```
    dimmerValue = analogRead(dimmerPin);           // 0–4095
```

```
    int pwmValue = map(dimmerValue, 0, 4095, 0, 255); // 8 bits
```

```
    analogWrite(ledPin, pwmValue);                 // Ajusta el brillo
```

```
} else {
```

```
    analogWrite(ledPin, 0);                         // Apaga el LED
```

```
}
```

- Cuando `lightOn == true`, leemos el potenciómetro y lo mapeamos a 0...255 para `analogWrite()`.
- Si está apagado, forzamos 0 en PWM para secar toda corriente al LED.

4.4. Impresión periódica en serie

```
if (millis() - lastPrintTime >= printInterval) {
```

```
    lastPrintTime = millis();
```

```
    Serial.print("Temp: ");
```

```
Serial.print(simulatedTemperatureC);
```

```
Serial.print("°C | LED: ");
```

```
Serial.print(lightOn ? "ON" : "OFF");
```

```
Serial.print(" | Dimmer: ");
```

```
Serial.println(dimmerValue);
```

```
}
```

- Solo mostramos un nuevo mensaje cada segundo (o el intervalo que definas), manteniendo la consola limpia.
- Mientras tanto, el bucle sigue respondiendo rápido al botón y al dimmer.

5- Implementar un prototipo del dispositivo antes mencionado con ESP32 y conectarlo a un Broker mediante Protocolo MQTT, visualizando en Smartphone o Tablet. En su defecto controlar y comunicar 3 dispositivos, sensores y/o actuadores, mediante el protocolo mencionado. Pueden usar Arduino, pero necesitan el módulo de comunicación a internet. El lenguaje de programación es a su elección, Python, C++, etc.

Se desarrolla un sistema de 3 componentes, que interactúan entre sí, emulando un sistema de sensorización, alarma y reacción.

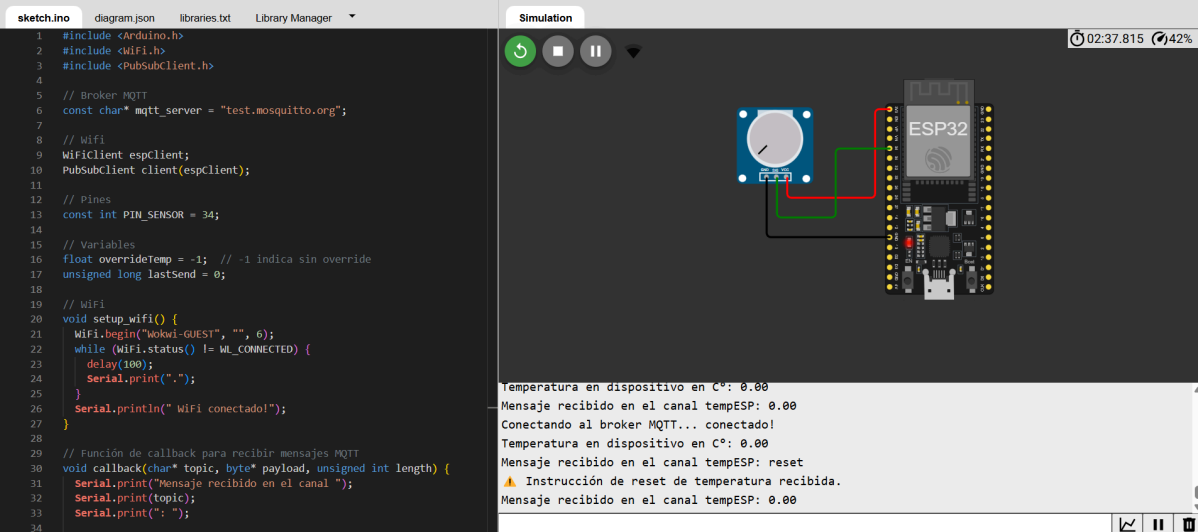
Creamos un dispositivo que funcionara como el sensor del trío, simulado en Wokwi. Un dispositivo, en físico que recibirá y alarmará cuando se cumplan ciertas condiciones, y un móvil. Que mediante una app podrá interactuar con el sistema, y funcionará a modo de elemento de reacción

Broker elegido: test.mosquitto.org

Topics:

- ☒ `tempESP`
- ☒ `AlarmaAltaTemperatura`

- **Dispositivo Sensor.** ESP32 simulado en wokwi, y corresponde simplemente a un potenciómetro que simula cambios de temperatura. Enviará la información (temperatura) cada 5 segundos a todos los elementos suscritos al topic que llamamos **tempESP**. Como el mensaje es informativo, y se envía cada cortos periodos, no es necesario asegurar la llegada del mensaje en cada envío, por lo que sale con QoS 0.



```

1 #include <Arduino.h>
2 #include <WiFi.h>
3 #include <PubSubClient.h>
4
5 // Broker MQTT
6 const char* mqtt_server = "test.mosquitto.org";
7
8 // Wifi
9 WiFiClient espClient;
10 PubSubClient client(espClient);
11
12 // Pines
13 const int PIN_SENSOR = 34;
14
15 // Variables
16 float overrideTemp = -1; // -1 indica sin override
17 unsigned long lastSend = 0;
18
19 // Wifi
20 void setup_wifi() {
21   WiFi.begin("wokwi-GUEST", "", 6);
22   while (WiFi.status() != WL_CONNECTED) {
23     delay(100);
24     Serial.print(".");
25   }
26   Serial.println(" WiFi conectado!");
27 }
28
29 // Función de callback para recibir mensajes MQTT
30 void callback(char* topic, byte* payload, unsigned int length) {
31   Serial.print("Mensaje recibido en el canal ");
32   Serial.print(topic);
33   Serial.print(": ");
34

```

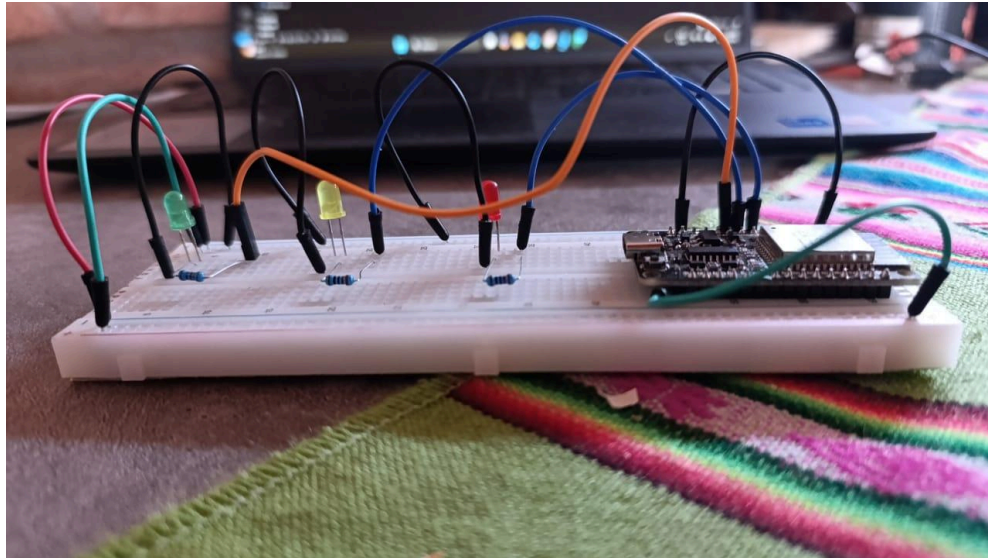
Temperatura en dispositivo en C°: 0.00
Mensaje recibido en el canal tempESP: 0.00
Conectando al broker MQTT... conectado!
Temperatura en dispositivo en C°: 0.00
Mensaje recibido en el canal tempESP: reset
⚠ Instrucción de reset de temperatura recibida.
Mensaje recibido en el canal tempESP: 0.00

El código y estructura en directorio: *C prototipo\Actividad N°3\Medidor Temperatura - Simulación Wokwi*

- **Dispositivo Alarma.** ESP32 físico con 3 leds. Este emulará un sistema de alarmas regular ante una variación. El led número 1, se encenderá durante 1 segundo, cada vez que el dispositivo reciba una lectura correspondiente al topic **tempESP**. Cuando dicho valor rompa el umbral establecido de 45°C , encenderá el led número 2 por 3 segundos. En un sistema real, podemos equiparar a una alarma de bajo nivel o advertencia. Y cuando el valor recibido corresponda a +65°C encenderá led 3, en lugar de led 2. Ahora además enviando un mensaje de alerta al dispositivo móvil. Como quisiéramos que este mensaje llegue obligatoriamente, lo enviamos con una calidad QoS 2 al topic **AlertaAltaTemperatura**.



El código y estructura en el directorio: *C prototipo\Actividad N°3\Alarma Temperatura*



- **Dispositivo de Reacción.** Este corresponde a una app del móvil (IoT MQTT móvil) que permite actuar y configurar gadgets con conexiones a canales, seguridad, calidad, etc. Este dispositivo estará con un gadget conectado al canal **tempESP**, recibiendo en tiempo real la información y estado de temperatura. Ante el mensaje de alarma recibido desde el Dispositivo Alarma (visible via un gadget log), se podrá presionar un botón de emergencia. Que envía en QoS 2 un mensaje de "reset" al Dispositivo Sensor, mediante el topic **AlertaAltaTemperatura**, reseteando el valor de temperatura a 0. Emulando lo que podría ser un la reacción de un operario o sistema automatizado.



6- Realizar el Dashboard y producir video de funcionamiento y presentación en *.ppt.

La presentación se encuentra alojada en: [Arquitectura-y-Conectividad/Dpresentacion/Actividad N°3/Pregunta 6.pptx at main · ISPC-Opalo/Arquitectura-y-Conectividad](#)

7- Imaginen que tienen una casa inteligente con sensores de temperatura, luces automáticas y una cafetera conectada. ¿Por qué sería conveniente usar el protocolo MQTT para que estos dispositivos se comuniquen entre sí? Mencionen al menos tres características del protocolo que justifiquen su respuesta.

¿Por qué es conveniente usar un broker en MQTT?

En el protocolo MQTT, el uso de un broker es fundamental para garantizar una comunicación eficiente, confiable y escalable entre dispositivos. *El broker actúa como intermediario entre los dispositivos que publican información* (como un sensor de movimiento) y los que se suscriben a esa información (como una luz, cámara o aplicación).

1. Desacoplamiento entre dispositivos

- Los dispositivos no necesitan conocerse entre sí ni establecer conexiones directas.
 - El sensor simplemente publica su mensaje en un tema, y el broker lo distribuye a todos los dispositivos suscritos.
 - Esto permite construir un sistema más modular, flexible y escalable.
-

2. Gestión centralizada y eficiente del tráfico

- El broker se encarga de filtrar, enrutar y distribuir los mensajes según los temas.
 - Puede manejar miles de dispositivos conectados simultáneamente.
 - Esto simplifica la arquitectura de red y mejora el rendimiento general del sistema.
-

3. Mayor confiabilidad y control

- El broker gestiona los niveles de calidad de servicio (QoS) y asegura que los mensajes se entreguen correctamente.
 - Puede almacenar mensajes retenidos para que los nuevos suscriptores reciban el último estado conocido.
 - Garantiza que no se pierdan eventos importantes, incluso si un dispositivo estaba desconectado temporalmente.
-

Conclusión:

El broker es un componente clave en MQTT porque:

- Centraliza la comunicación entre dispositivos.
- Mejora la eficiencia y confiabilidad del sistema.
- Permite una arquitectura de red más ordenada, flexible y escalable.

En sistemas de IoT como un sensor de movimiento que controla otros dispositivos, el broker permite que todo funcione de manera coordinada y autónoma.

8- Supongamos que un sensor de movimiento instalado en el pasillo publica un mensaje cuando detecta movimiento. Las luces del pasillo están configuradas para encenderse cuando reciben ese mensaje. Explicar cómo funciona esta comunicación usando el modelo publicador/suscriptor de MQTT, e indicar cuál es el papel del broker.

Encendido de luces por detección de movimiento usando el protocolo MQTT.

Comunicación entre el sensor de movimiento y las luces.

Cuando el sensor de movimiento, ubicado en el pasillo, detecta la presencia de una persona, actúa como cliente MQTT y se comporta como publicador. Este sensor establece una conexión con el broker MQTT a través de una conexión TCP/IP.

Establecimiento de la conexión

Para conectarse, el cliente envía un mensaje CONNECT, que contiene:

- client-id (identificador del cliente),
- nombre de usuario y contraseña (si el broker lo requiere).

El broker responde con un mensaje CONNACK, indicando si la conexión fue aceptada o rechazada.

Por defecto, MQTT utiliza el puerto 1883 para conexiones sin cifrado.

Publicación del mensaje

Una vez establecida la conexión, el sensor envía un mensaje PUBLISH que incluye:

- Un topic o tema, por ejemplo: pasillo/movimiento.
- Un payload o contenido del mensaje, por ejemplo: "ON" para indicar que se detectó movimiento.

El broker recibe ese mensaje y aplica un filtrado por topic para identificar qué clientes están suscritos a ese tema y reenviarles el contenido.

Recepción y acción de las luces

Las luces del pasillo, también configuradas como clientes MQTT, enviaron previamente un mensaje SUBSCRIBE al broker, indicando su interés en recibir mensajes del topic pasillo/movimiento.

Cuando el broker recibe un mensaje con este topic (por ejemplo, "ON"), lo reenvía automáticamente a todas las luces que están suscritas. Al recibir el mensaje, las luces ejecutan la acción previamente programada: *encenderse*.

Rol del broker

- El broker MQTT cumple un rol central en este sistema. Se encarga de:
- Recibir los mensajes publicados por los clientes.
- Filtrar los mensajes según el topic.

Reenviar los mensajes solo a los clientes que estén suscritos al topic correspondiente.

Este diseño desacopla completamente los dispositivos emisores de los receptores, brindando flexibilidad y escalabilidad al sistema.
