

UNIVERSIDAD NACIONAL MAYOR DE SAN MARCOS

Facultad de Ingeniería de Sistemas e Informática

Escuela Profesional de Ingeniería de Sistemas



CONTROL Y MONITOREO REMOTO DE LA TEMPERATURA, HUMEDAD Y CO DE UN HORNO INDUSTRIAL

Docente: Yessica Rosas Cueva

Asignatura: Internet de las Cosas

Sección 1

Grupo 2

Integrantes:

Bravo Chuquillanqui Jhamil Rodrigo	20200159
Romero Ruiz José Daniel	20200208
Olaechea Saavedra, Leonardo Cashiel	20200052
Alvarado Flores Sebastian Paulo	20200149
Zapata Sanchez Renzo Marcelo	18190172

LIMA - PERÚ

2023 - 0

ÍNDICE

1. Introducción	3
1.1. Revisión del estado del arte	3
1.2. Planteamiento del problema	6
1.3. ODS enfocado al Proyecto	7
1.4. Objetivos	7
2. Marco teórico	7
2.1. Sensores	7
2.2. Requerimientos del Horno Industrial	8
2.3. IoT	8
2.4. Node Red	8
2.5. Base de datos	9
2.6. Broker emqx	9
2.7. Arduino	9
2.8. Docker	10
3. Componentes del sistema	11
3.1. ESP32	11
3.2. Sensor DTH11	12
3.3. Sensor MQ9	13
3.4. Sensor FC-51	14
3.5. Relé HW-482	15
3.6. Foco Incandescente	15
3.7. LCD 16X2	16
3.8. Resistencia 1K	17
3.9. Protoboard	18
4. Implementación del sistema	19
5. Resultados	35
6. Conclusiones	37
7. Bibliografía	37
8. Anexos	38

1. Introducción

1.1. Revisión del estado del arte

- Registro automático de temperatura y activación de alarmas en cuatro zonas de un horno de extrusión

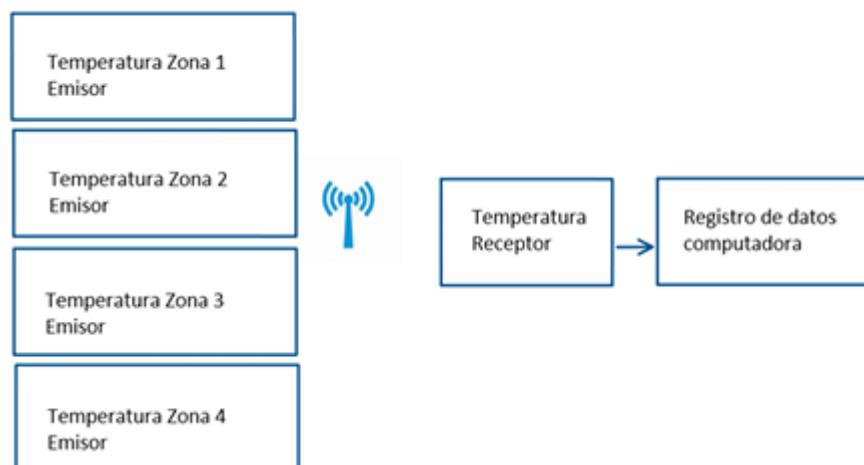
Rascon Madrigal, L. H. (2022). Registro automático de temperatura y activación de alarmas en cuatro zonas de un horno de extrusión.

Instituto de Ingeniería y Tecnología.

Objetivo: Desarrollar un sistema de registro automático de temperatura y activación de alarmas en cuatro zonas de un horno de extrusión para verificar que la temperatura se encuentre dentro de las especificaciones del cliente.

Funcionamiento: Luego de armar el proyecto en este se encuentra un sistema de medida y registro de temperaturas inalámbrico en 4 zonas de un horno, basado en el protocolo LoRa y un receptor, que colecta y almacena las lecturas de temperatura.

A continuación les dejamos un esquema que ejemplifica mejor el funcionamiento:



Conclusiones: El desarrollo del proyecto lleva un gran avance, este registra simultáneamente la temperatura de 2 estaciones, el receptor recibe y almacena la temperatura, sin embargo hubo un problema con los otros módulos LoRa.

- **Diseño e implementación de prototipo de sistema electrónico monitoreado en tiempo real para programa de producción de fábrica de vidrio**

Bravo Cedeño, G. S. (2020). Diseño e implementación de prototipo de sistema electrónico monitoreado en tiempo real para programa de producción de fábrica de vidrio (Doctoral dissertation, Universidad de Guayaquil. Facultad de Ciencias Matemáticas y Físicas. Carrera de Ingeniería en Networking y Telecomunicaciones).

Objetivos: Diseñar e implementar un prototipo de sistema electrónico para monitorear y controlar en tiempo real la eficiencia productiva de una fábrica de vidrio.

Funcionamiento: Se colocan los sensores a lo largo de la banda transportadora lo cual permite tener una idea clara de la toma de datos con un objeto en movimiento. Además ya se encuentra preestablecido los rangos operacionales por medio del software..

Conclusiones: Se concluye que el proyecto debe ser monitoreado a tiempo real para responder de forma inmediata a los problemas con la producción defectuosa de los envases de vidrio.

- **Sistema inteligente de alerta de incendios o fugas de gas licuado de petróleo para viviendas de adultos mayores utilizando el protocolo LORA**

Paspuel Pozo, T. L. (2022). Sistema inteligente de alerta de incendios o fugas de gas licuado de petróleo para viviendas de adultos mayores utilizando el protocolo Lora (Bachelor's thesis).

Objetivos: Implementar un sistema inteligente de alerta de incendio y GLP basado en el protocolo LoRa para disminuir muertes por inhalación del GLP e incendios en viviendas de adultos mayores.

Funcionamiento: El sistema está construido en 4 bloques que permiten su funcionalidad, en el primer bloque está el nodo colector que se encarga de censar el medio con la utilización de sensores MQ-2 y MQ-5 , en el segundo bloque por el Gateway que recibe los datos del nodo colector , los procesa y envía mediante el protocolo MQTT al siguiente bloque

denominado Servidor IOT ThingSpeak y en caso de exceder el valor del umbral avisa al usuario mediante un correo electrónico y el último bloque viene siendo la presentación de datos al usuario.

Conclusiones: La utilización de la tecnología en este proyecto contribuye a la monitorización de viviendas de adultos mayores, ayudando a detectar presencia de GLP o humo y generando una señal de alerta que permita a la persona tutor actuar a tiempo, evitando accidentes graves que pueden ocasionar hasta la muerte.

- **Sistema automatizado para la identificación de defectos y la clasificación de baldosas cerámicas en la salida del horno**

Espinoza Dueñas, B. (2019). Sistema automatizado para la identificación de defectos y la clasificación de baldosas cerámicas en la salida del horno.

Objetivos: Implementar un sistema automatizado para la identificación de defectos y la clasificación de baldosas cocidas.

Funcionamiento: Se inicia cuando la baldosa sale del horno, luego pasa a la línea de transporte e ingresa al equipo de clasificación automático, las tres cámaras capturan imágenes de la pieza, cuando es detectado el defecto mediante la gestión que realiza el software del equipo a la imagen, envía una señal a la top jet (impresora de tinta, ver Figura 19) para imprimir una marca en la pieza. Antes del apilado de piezas un sensor detecta la marca; si la marca se encuentra al final de la baldosa envía la señal a un mecanismo para que descarte la pieza dentro de un contenedor (ver Figura 10) y si la marca se ubica al inicio de la pieza se encajona como producto de segunda calidad.

Conclusiones:

Se logró mejorar la capacidad de detección de los defectos. Al evaluar los tres principales defectos de estudio, se obtuvo una precisión con el sistema automatizado de 1.53% respecto al 1.22% del sistema manual. Se comprobó que el sistema automatizado en promedio tiene 0.31% de mejora en la detección de los tres defectos en estudio con respecto al sistema manual.

1.2. Planteamiento del problema

En una fábrica de producción de vidrio, se enfrentan a un problema recurrente en la calidad de los productos que se obtienen del horno de fusión. Aunque existen hornos industriales automatizados, estos son muy costosos, en cambio, en los hornos antiguos, los trabajadores encargados de operar el horno deben monitorear manualmente la temperatura y otros parámetros de funcionamiento para asegurarse de que se cumpla con el proceso de fusión adecuado. Sin embargo, esto no es suficiente para garantizar la calidad constante del producto. El proceso de fusión del vidrio es muy delicado y puede variar dependiendo de diferentes factores, como la humedad del aire, la cantidad de vidrio en el horno y otros factores como la cantidad de monóxido emitido. Además, los trabajadores tienen que depender de su experiencia y habilidades para ajustar manualmente el horno y realizar cambios en la temperatura y otros parámetros durante el proceso de fusión, lo que puede llevar a errores humanos y resultados inconsistentes. Para garantizar la calidad del producto y reducir el desperdicio, se necesita una solución que permita monitorear y controlar el horno de manera más precisa y eficiente. Una posible solución es utilizar la tecnología IoT para recopilar datos en tiempo real sobre la temperatura y otros parámetros relevantes, y ajustar automáticamente el horno según los parámetros de producción específicos. Con el control y monitoreo del horno a través de IoT, se espera reducir los errores humanos, aumentar la eficiencia del proceso de fusión, mejorar la calidad del producto y reducir los costos de producción. Además, los trabajadores podrán supervisar el proceso de fusión desde cualquier lugar a través de un dispositivo móvil, lo que les permitirá realizar ajustes rápidos y precisos en caso de que se produzcan desviaciones en los parámetros de producción. Esto permitirá que la fábrica tenga mayor control sobre el proceso de fusión, lo que se traducirá en una mejor calidad y consistencia del vidrio que se produce.

1.3. ODS enfocado al Proyecto

El siguiente ODS es en el que está enfocado nuestro proyecto. ODS 12: Garantizar el consumo y producción sostenibles. El uso de sensores y dispositivos IoT en un horno puede ayudar a monitorizar la eficiencia y sostenibilidad del proceso de producción, lo que podría reducir el impacto ambiental y mejorar la eficiencia en la producción.

1.4. Objetivos

- Implementar un sistema de control tipo ON-OFF y monitoreo de temperatura de un horno haciendo uso del ESP32 como controlador, el sensor DTH11 como sensor y un foco incandescente como actuador.
- Configurar y desplegar los servicios de broker mqtt, base de datos y dashboard necesarios para el funcionamiento de la aplicación.
- Diseñar e implementar un dashboard de monitoreo en tiempo real de las variables medidas en el sistema haciendo uso de node-red.

2. Marco teórico

2.1. Sensores

Los sensores son dispositivos electrónicos que permiten la captura de información sobre el entorno en el que se encuentran, tales como la temperatura, la humedad, la presión, la luz, entre otros. En el contexto de IoT, los sensores son elementos fundamentales para obtener datos que pueden ser utilizados para tomar decisiones y mejorar procesos.

Los sensores en IoT pueden funcionar de diferentes maneras, existen muchos tipos de sensores disponibles, se comunican con otros dispositivos a través de diferentes protocolos, tienen un bajo consumo de energía, deben ser calibrados de forma regular y deben ser integrados en sistemas más grandes.

2.2. Requerimientos del Horno Industrial

Un sistema IoT para un horno industrial debe contar con sensores precisos y confiables que permitan medir diferentes parámetros del horno, un sistema de control y monitoreo que permita recopilar y procesar los datos en tiempo real, conectividad a Internet confiable y segura, y una base de datos escalable y eficiente para almacenar y procesar los datos generados por el horno. Con estos requerimientos, se puede garantizar el correcto funcionamiento del horno y asegurar la calidad del producto final.

2.3. IoT

El Internet de las cosas (IoT) es una red de objetos físicos, dispositivos y otros objetos cotidianos que se conectan a internet y se comunican entre sí sin la necesidad de intervención humana directa. Estos dispositivos tienen sensores y actuadores incorporados que les permiten recopilar información, analizarla y tomar decisiones basadas en ella.

El IoT tiene el potencial de revolucionar muchos aspectos de la vida cotidiana y los negocios. Al permitir una mayor automatización y eficiencia en los procesos y servicios, el IoT puede mejorar la productividad, reducir costos, mejorar la seguridad y la calidad de vida, y permitir la creación de nuevas oportunidades de negocio.

2.4. Node Red

Node-RED es una herramienta de programación visual que permite la creación de aplicaciones IoT y de automatización mediante la interconexión de diferentes nodos de software.

El funcionamiento de Node-RED se basa en la creación de flujos de trabajo visuales, donde cada nodo representa una función específica.

Este es compatible con una amplia variedad de dispositivos y servicios de IoT, y cuenta con una amplia biblioteca de nodos preconstruidos que facilitan la integración con diferentes plataformas y servicios en la nube.

2.5. Base de datos

Las bases de datos son un componente clave en los sistemas IoT, ya que se utilizan para almacenar y procesar grandes cantidades de datos generados por los dispositivos IoT. La cantidad de datos que se generan en los sistemas IoT es enorme y diversa, lo que hace que el diseño de una base de datos eficiente y escalable sea un desafío importante en la implementación de soluciones IoT.

2.6. Broker emqx

EMQ X es un broker de mensajería de código abierto que se utiliza para implementar sistemas de mensajería IoT y M2M. Es un servidor de intermediación que recibe mensajes de dispositivos y los envía a los dispositivos de destino.

El broker EMQ X está diseñado para manejar grandes volúmenes de mensajes de manera eficiente y escalable, lo que lo hace ideal para su uso en sistemas IoT y M2M. Puede procesar millones de mensajes por segundo y soporta una gran cantidad de conexiones concurrentes.

2.7. Arduino

Arduino es una plataforma de hardware libre y de código abierto que se utiliza ampliamente en proyectos de IoT (Internet de las cosas). La plataforma Arduino se puede utilizar para crear dispositivos que recopilan datos de sensores y los envían a la nube para su análisis y almacenamiento. También se puede utilizar para controlar dispositivos y actuadores remotamente.

La combinación de la plataforma Arduino y el IoT ofrece una amplia gama de posibilidades para la automatización y el control de dispositivos, desde hogares inteligentes hasta ciudades inteligentes. La plataforma Arduino se puede utilizar para crear dispositivos que recopilan y transmiten datos de sensores en tiempo real, como la temperatura, la humedad, la presión y la calidad del aire.

2.8. Docker

Docker es una plataforma de virtualización que permite a los desarrolladores crear, implementar y ejecutar aplicaciones en contenedores. Los contenedores son una forma ligera y portátil de encapsular una aplicación y todas sus dependencias, lo que hace que sea fácil moverla entre entornos de desarrollo, pruebas y producción.

En el contexto de IoT, Docker se utiliza para crear contenedores que ejecutan aplicaciones en dispositivos IoT. Los dispositivos IoT a menudo tienen recursos limitados, como memoria y potencia de procesamiento, y la virtualización mediante Docker permite a los desarrolladores ejecutar aplicaciones en estos dispositivos de manera más eficiente.

3. Componentes del sistema

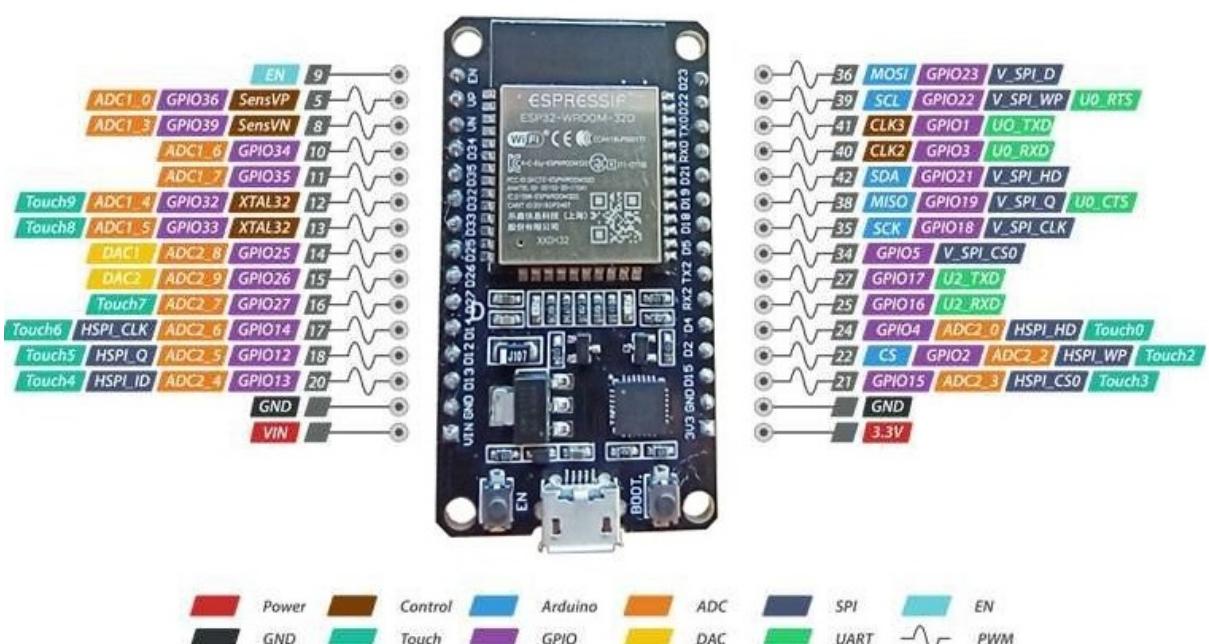
3.1. ESP32

El ESP32 es un microcontrolador de bajo costo y alto rendimiento fabricado por Espressif Systems (Espressif Systems, 2021).

Es una evolución del ESP8266, que ha ganado popularidad en los últimos años debido a su capacidad de conectividad Wi-Fi. El ESP32 se basa en una arquitectura de doble núcleo Xtensa LX6, que proporciona una velocidad de reloj de hasta 240 MHz. Además de la conectividad Wi-Fi, el ESP32 también cuenta con Bluetooth LE, lo que lo hace ideal para aplicaciones de IoT (Random Nerd Tutorials, 2021).

Características principales:

- Procesador Tensilica Xtensa 32bits LX6 hasta 240MHz.
- Wi-Fi: 802.11b/g/n/e/i (802.11n @ 2.4 Ghz hasta 150 Mbit/s).
- Bluetooth: v4.2 BR/EDR y bluetooth Low Energy (BLE).
- Rom:448 KiB.
- SRAM: 520 KiB.
- RTC slow SRAM: 8 KiB.
- RTC fast SRAM: 8 KiB.
- Seguridad tipo IEEE 802.11, WFA, WPA/WPA2 y WAPI.
- Voltaje de trabajo 3.3VDC.
- Energía y datos via conector microUSB 5VDC.



3.2. Sensor DTH11

El DHT11 es un sensor de temperatura y humedad que utiliza un sensor capacitivo de humedad y un termistor para medir el aire circundante, y muestra los datos mediante una señal digital en el pin de datos. Es un sensor de bajo costo y fácil de usar, que proporciona mediciones de humedad relativa y temperatura con una precisión moderada.

Características físicas

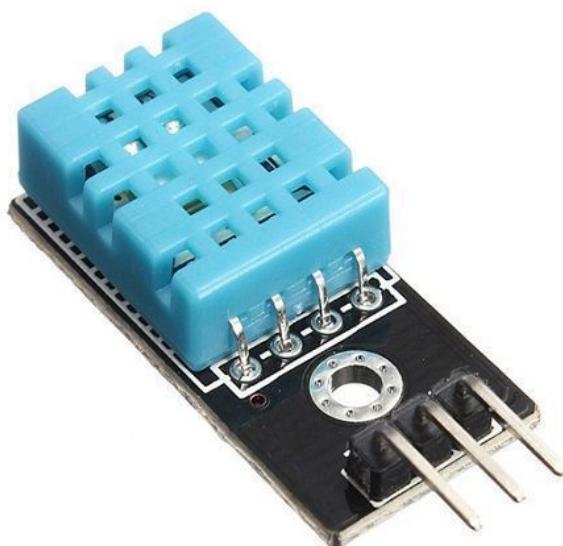
- Dimensiones: 23 mm x 12 mm x 5 mm
- Peso: 1 gramo
- Pin 1-Alimentación 3.5 a 5.5V DC
- Pin 2-Datos
- Pin 3-NC (No usado)
- Pin 4-GND (Tierra)
- Corriente de operación: 2.5 mA (máximo)

Parámetros de medición

- Rango de medición de temperatura: 0°C a 50°C
- Precisión de la medición de temperatura: $\pm 2^\circ\text{C}$
- Rango de medición de humedad: 20% a 90% HR
- Precisión de la medición de humedad: $\pm 5\%$ HR

Tipo de señal

- Emplea una señal digital de salida, el cual consiste en un tren de pulsos digitales que representan la humedad y las temperaturas medidas
- Es de tipo serie transmitido mediante protocolo propietario de un solo cable



3.3. Sensor MQ9

Es un sensor de gas que se utiliza para detectar la presencia de monóxido de carbono (CO), metano (CH₄) y otros gases inflamables en el aire. Este sensor funciona mediante la medición de la resistencia eléctrica de un elemento sensor sensible a los gases en el aire.

Características física

- Dimensiones: 32 mm x 22 mm x 27 mm
- Peso: 5 gramos
- alimentación: 5V DC
- Corriente de operación: 150 mA
- Temperatura de operación: -10°C a 50°C

Parámetros de medición

- Rango de medición de monóxido de carbono (CO): 10 a 1000 ppm (partes por millón)
- Sensibilidad de monóxido de carbono (CO): 1.5% / ppm
- Rango de medición de gases inflamables: 100 a 10,000 ppm
- Sensibilidad de gases inflamables: 1% / ppm
- Tiempo de respuesta: < 10 segundos
- Tiempo de recuperación: < 30 segundos

Tipo de señal

- El MQ-9 genera una señal analógica de voltaje que es proporcional a la concentración de gas medida en el ambiente. La señal de salida varía en un rango de 0 a 5 voltios y se puede leer utilizando un microcontrolador o un circuito de acondicionamiento de señal.



3.4. Sensor FC-51

El sensor FC-51 es un tipo de sensor de detección de obstáculos o barrera de infrarrojos. Este consta de un emisor infrarrojo y un receptor infrarrojo. Además, se le conoce como un sensor de seguimiento de línea ya que puede detectar líneas blancas o negras en una superficie. El sensor emite un haz de luz infrarroja y el receptor detecta la cantidad de luz reflejada por la superficie.

Características físicas:

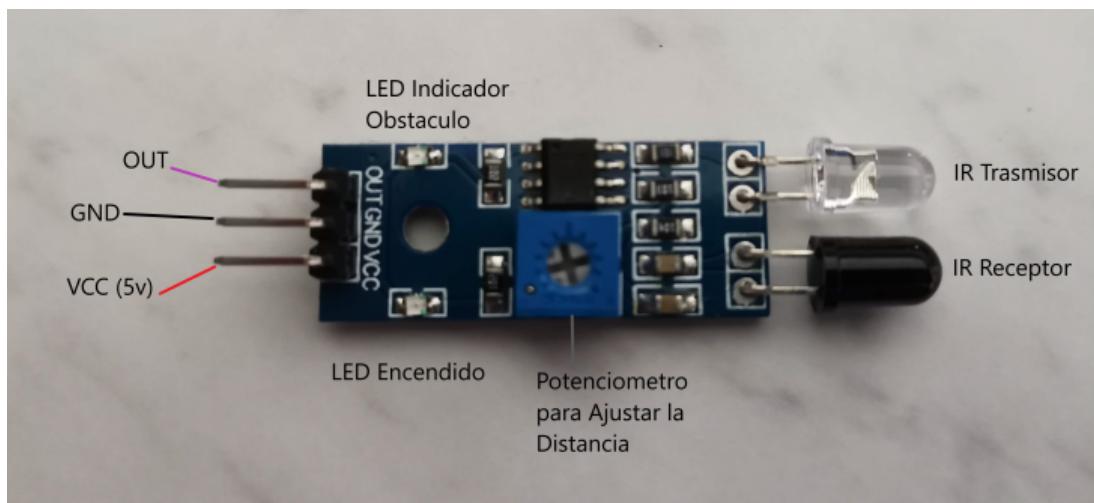
- Dimensiones: 3.2 cm x 1.4 cm x 0.8 cm
- Peso: 3 gramos
- Voltaje de alimentación: 3.3V – 5V
- Voltaje de salida digital : 5V
- VCC: Voltaje de alimentación
- OUT: Salida de tensión digital (0,1)
- GND: Tierra

Parámetros de medición:

- Distancia de detección: 2cm - 3cm
- Ángulo de detección: 35 grados

Tipo de señal:

- Señal digital: El sensor produce una salida digital que indica si se detectó o no un obstáculo o línea. La salida es de 0V (bajo) si se detecta un obstáculo o línea, y de 5V (alto) si no se detecta nada.



3.5. Relé HW-482

Un módulo relé es un componente electrónico que permite el control de corriente eléctrica de alta potencia con una señal de corriente eléctrica de baja potencia. Se trata de un interruptor eléctrico controlado que se activa o desactiva mediante una señal de control de bajo voltaje y corriente, como una señal proveniente de un microcontrolador o un interruptor mecánico.

Características físicas::

- Módulo de relé 5v
- Señal del control TTL 5v a 12v
- Voltaje máximo en AC 220v



3.6. Foco Incandescente

Un foco incandescente produce luz a través de un filamento de tungsteno que se calienta mediante la electricidad, provocando su incandescencia y la emisión de radiación electromagnética en el espectro visible. Aunque han sido utilizados por más de un siglo como una de las formas más comunes de iluminación artificial, su baja eficiencia energética ha llevado a la popularidad de alternativas más eficientes como las bombillas LED y CFL.

Características físicas:

- Tipo de base: Los focos incandescentes pueden tener diferentes tipos de bases, como E26, E12, GU10, entre otros.
- Potencia: La potencia de un foco incandescente se mide en vatios y puede variar de 15 a 150 vatios o más.
- Voltaje: El voltaje de un foco incandescente suele ser de 120V o 240V, dependiendo de la región o país.
- Eficiencia energética: La eficiencia energética de un foco incandescente es baja, generalmente en torno al 10%, lo que significa que solo convierte el 10% de la energía en luz y el resto se pierde en forma de calor.



3.7. LCD 16X2

Un LCD 16x2 es una pantalla de cristal líquido que muestra hasta 16 caracteres en dos líneas. Es comúnmente utilizado en dispositivos electrónicos, automatización del hogar, control y monitoreo. A pesar de su tamaño compacto, muestra información clara y legible y puede ser controlado mediante señales de un microcontrolador o dispositivo de control. El LCD 16x2 es una alternativa económica a las pantallas más grandes y costosas, ideal para proyectos de bajo costo y espacio limitado.

Características físicas:

- Retroiluminación: La mayoría de las pantallas LCD 16x2 tienen retroiluminación LED para mejorar la visibilidad de la pantalla en condiciones de baja luz.
- Consumo de energía: Las pantallas LCD 16x2 consumen poca energía, generalmente entre 1 y 2 vatios.
- Tipo de pantalla: Son pantallas de cristal líquido (LCD) de matriz de puntos.

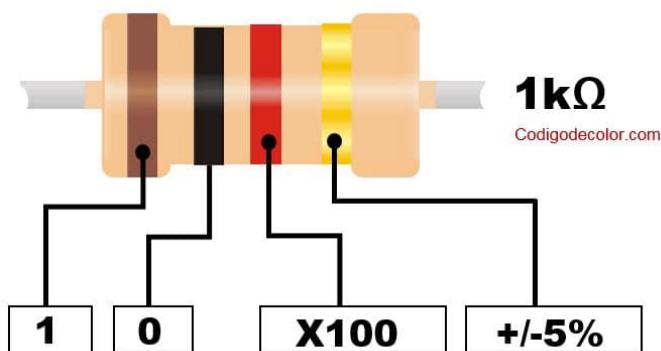


3.8. Resistencia 1K

La resistencia 1k es una resistencia eléctrica cuyo valor de resistencia es de 1000 ohmios (1 kiloohmio). Empleado comúnmente en circuitos eléctricos para limitar la corriente o para ajustar el voltaje en un punto específico del circuito.

Características:

- Valor nominal: 1 kiloohmio (1000 ohmios).
- Tolerancia: 5%
- Potencia máxima: 1/4 vatio (0.25 vatios)
- Coeficiente de temperatura: 200 partes por millón por grado Celsius (ppm/°C).
- Voltaje máximo: alrededor de 200-250 voltios.

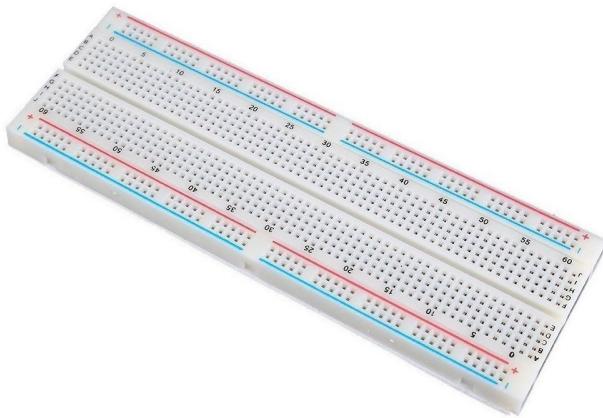


3.9. Protoboard

Una protoboard, también conocida como placa de prototipado, es una placa de circuito impreso que se utiliza para probar y construir circuitos electrónicos. La protoboard tiene una serie de hileras y columnas de contactos conductores que se pueden conectar entre sí mediante puentes o componentes electrónicos para crear un circuito temporal.

Características físicas:

- 830 puntos
- Color blanco
- Plástico ABS
- Compatible con cualquier componente o cable de 20-29 AWG
- Longitud de 16.5cm
- Ancho de 5.5 cm
- Altura de 1 cm



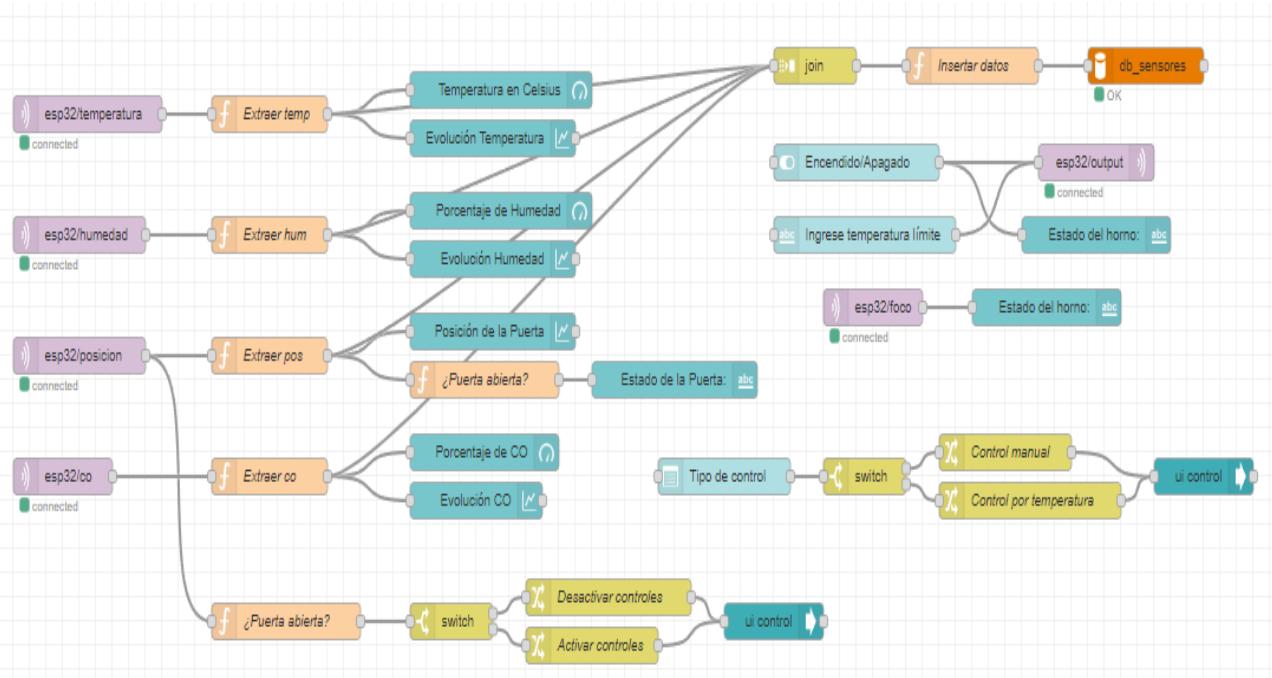
4. Implementación del sistema

Dentro del Red Node vamos a tener un nodo que va a tener la conexión con nuestra base de datos que en este caso estamos usando mysql, se guardan los datos dentro de la tabla esp32data dentro de la base de datos llamada db_sensores, en la imagen podemos ver la data recopilada de cada uno de los tres sensores en cuatro mediciones, como podemos ver los datos son recolectados cada segundo

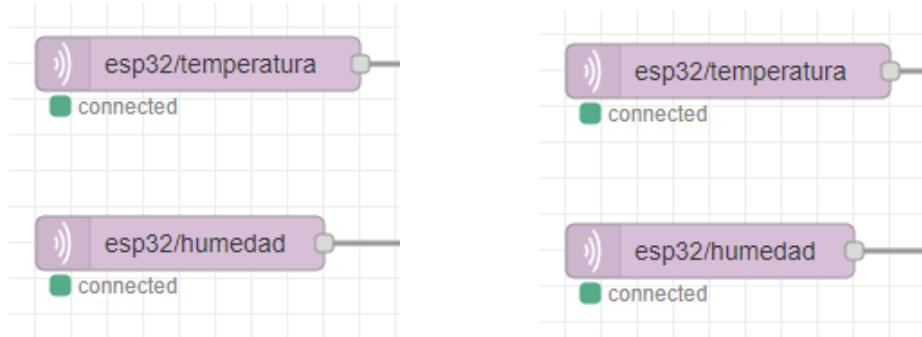
The screenshot shows the phpMyAdmin interface for the database 'db_sensores'. The left sidebar lists databases like bd_parcial, db_sensores, and esp32data. The main area displays the 'esp32data' table with the following columns: ID, FECHA, DEVICE, TEMPERATURA, HUMEDAD, MONOXIDO, and POSICION. The data shows 12 rows of sensor readings from an ESP32 device at various times on March 9, 2023.

	ID	FECHA	DEVICE	TEMPERATURA	HUMEDAD	MONOXIDO	POSICION
	26	2023-03-09 00:50:08	ESP32	28.8	57	7	1
	27	2023-03-09 00:50:09	ESP32	28.8	57	7	1
	28	2023-03-09 00:50:10	ESP32	28.8	57	7	1
	29	2023-03-09 00:50:12	ESP32	28.8	57	7	1
	30	2023-03-09 00:50:13	ESP32	28.8	57	7	1
	31	2023-03-09 00:50:14	ESP32	28.8	57	7	1
	32	2023-03-09 00:50:15	ESP32	28.8	57	7	1
	33	2023-03-09 00:50:16	ESP32	28.8	57	7	1
	34	2023-03-09 00:50:17	ESP32	28.8	57	7	1
	35	2023-03-09 00:50:18	ESP32	28.8	57	7	1
	36	2023-03-09 00:50:19	ESP32	28.8	57	7	1

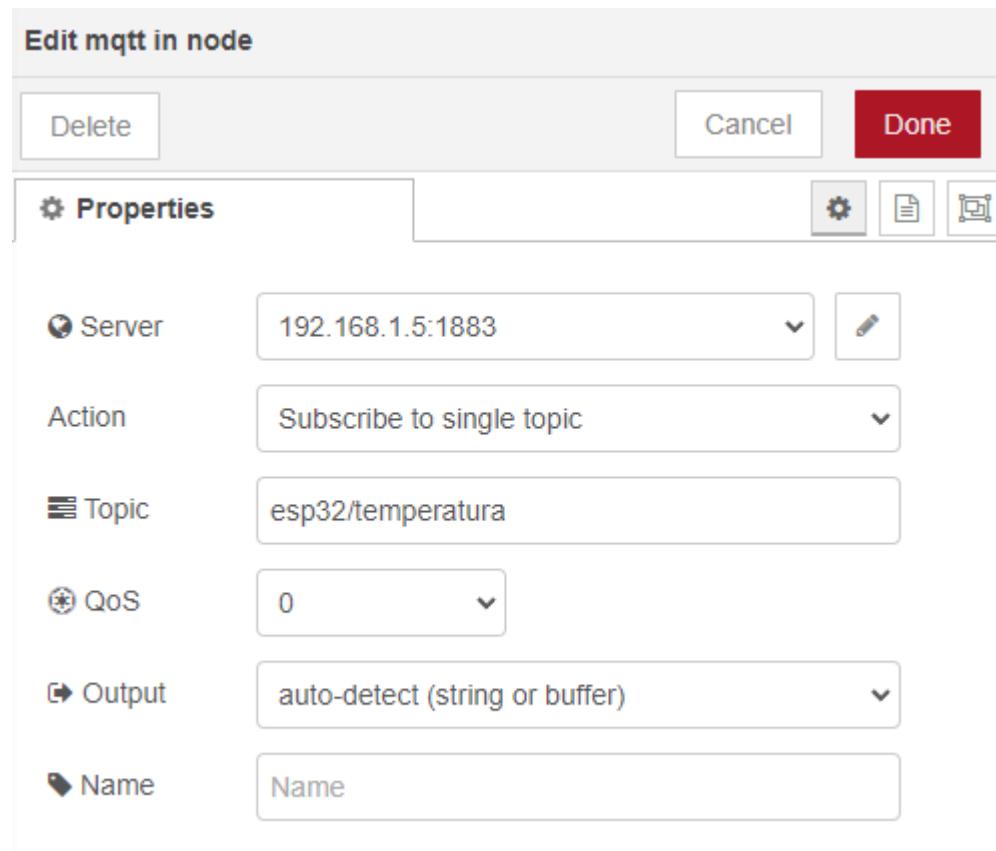
Esta es la construcción de los nodos del Node Red para el envío de datos, a continuación se verá el funcionamiento de cada parte del diagrama



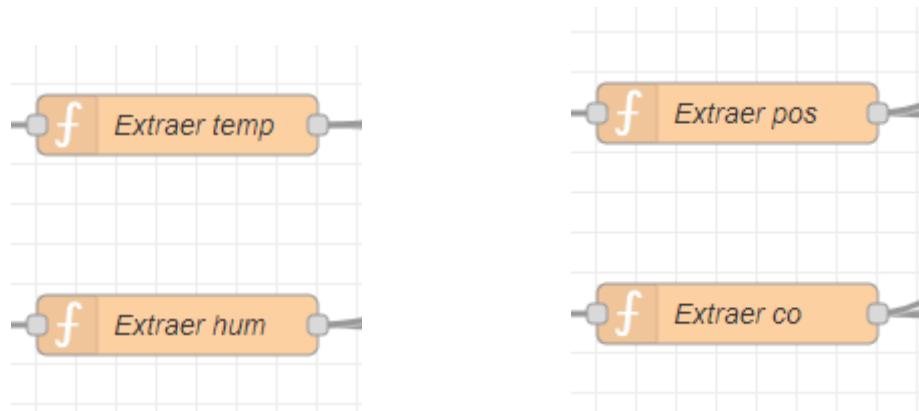
Estos cuatro nodos son llamados mqtt in, esta clase de nodo se utiliza para suscribirse a un tema (topic) específico en una red MQTT y recibir los mensajes publicados en ese tema. Este nodo puede configurarse para filtrar los mensajes según el contenido o el tipo de datos que contienen, en nuestro caso estos son los que van a recibir los datos del broker



Dentro de cada una definimos el servidor de donde se van a recibir los datos



Estas cuatro son funciones intermedias, son de clase función este es un tipo de nodo que permite escribir código personalizado en JavaScript para manipular los mensajes que se transmiten a través de la plataforma



Dentro de estas, transformamos el mensaje de los mqtt in a un objeto JSON, y de este objeto JSON extraemos el valor numérico y lo retornamos al siguiente nodo

Edit function node

Delete Cancel Done

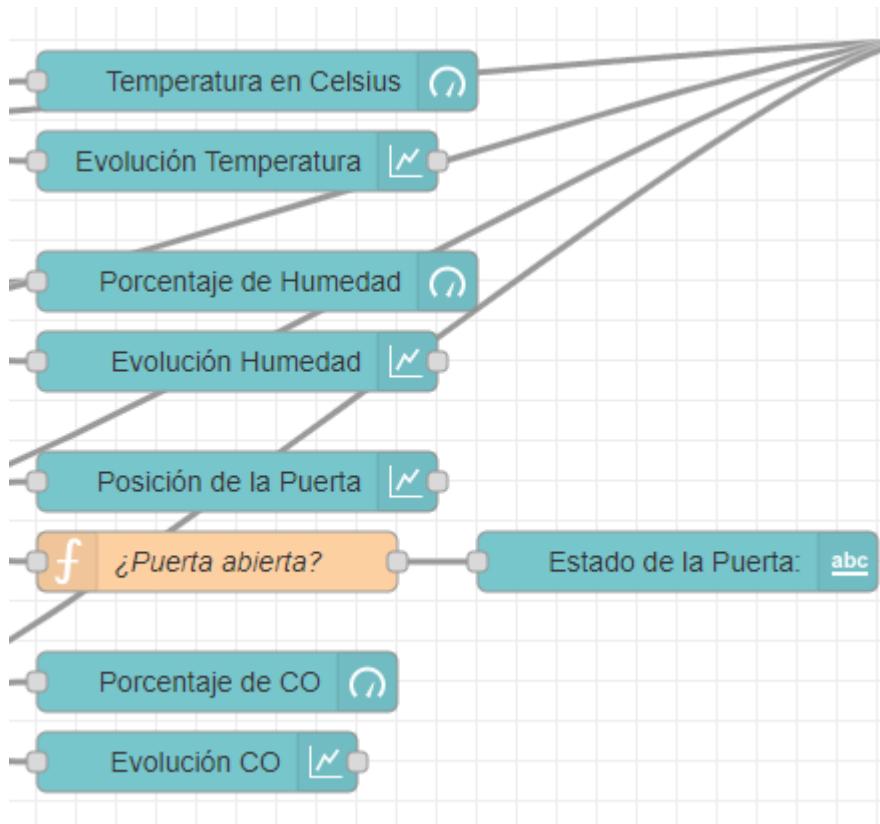
Properties

Name: Extraer temp

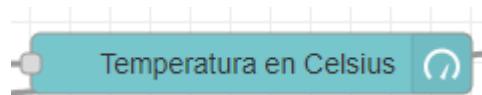
Setup On Start **On Message** On Stop

```
1 // Parsear el payload como un objeto JSON
2 var payloadObj = JSON.parse(msg.payload);
3
4 // Extraer la temperatura del objeto
5 var temperatura = payloadObj.temperatura;
6
7 // Crear un nuevo mensaje con la temperatura extraída
8 msg.payload = temperatura;
9
10 // Devolver el mensaje modificado
11 return msg;
```

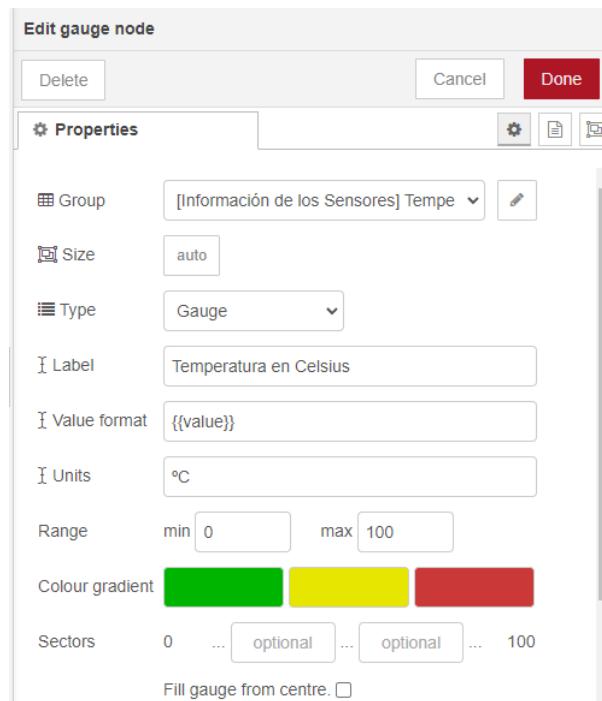
Los valores retornados se guardan en estos nodos de clase gráficos, este es un tipo de nodo que permite visualizar datos en forma de gráficos en tiempo real, utilizando bibliotecas de gráficos de terceros, como Chart.js o Highcharts, para generar gráficos de barras, líneas, áreas y otros tipos de gráficos a partir de datos que se reciben a través del flujo, en nuestro caso estamos usando las bibliotecas Dashboard, vemos que hay 2 tipos en los nodos celestes uno viene siendo gauge y el otro chart, el primero permite visualizar datos en forma de medidor o indicador de nivel, mientras que el segundo permite visualizar datos en forma de gráfico de líneas o de barras



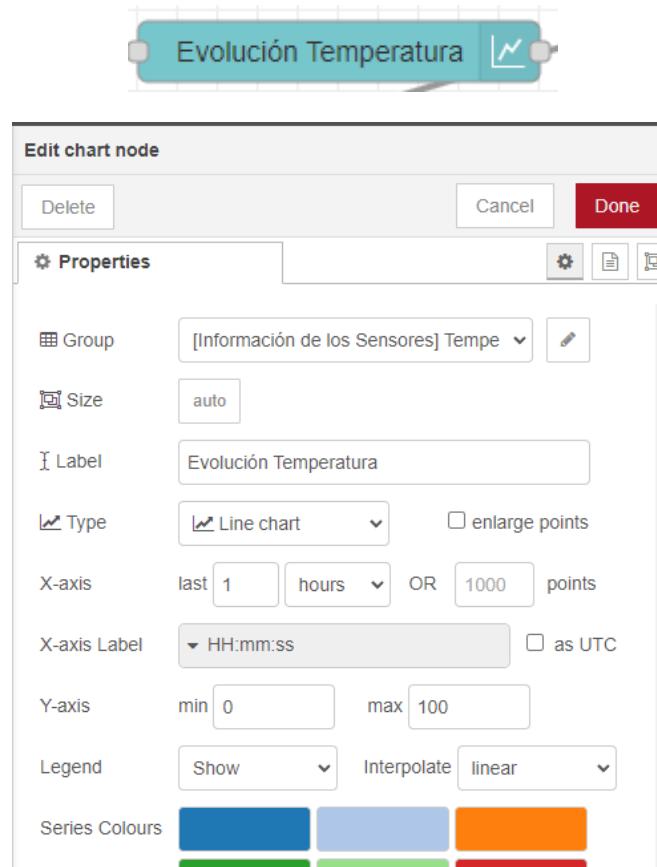
En la configuración de los nodos gráficos se les asigna dentro de un grupo específico



La interfaz para editar el node de tipo gauge llamado “Temperatura en Celsius”



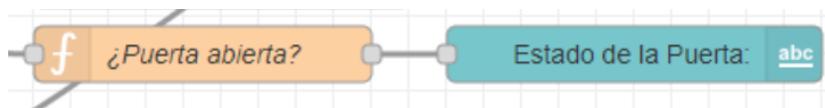
La interfaz para editar el node de tipo gauge llamado “Evolución Temperatura”



Aquí podemos ver todos los grupos y sus elementos

The screenshot shows the Node-RED interface with the 'Layout' tab selected. The 'Tabs & Links' panel displays a hierarchical structure of sensor information. The 'Información de los Sensores' tab is expanded, revealing five main categories: Temperatura, Humedad, Monóxido de Carbono, Posición, and Switch. Each category contains sub-items such as 'Temperatura en Celsius', 'Evolución Temperatura', 'Porcentaje de Humedad', 'Evolución Humedad', 'Porcentaje de CO', 'Evolución CO', 'Posición de la Puerta', 'Estado de la Puerta:', and 'Encendido/Apagado'.

También vemos que hay otra función que sigue de posición que se llama “¿Puerta abierta?” esta va a recibir un valor ya sea 1 si la puerta está abierta o 0 si está cerrada, devolviendonos un string que se transmite al siguiente nodo que es de clase texto que nos muestra el label: “Estado de la puerta” seguido del valor que llega de la función anterior



Edit function node

Delete Cancel Done

Properties

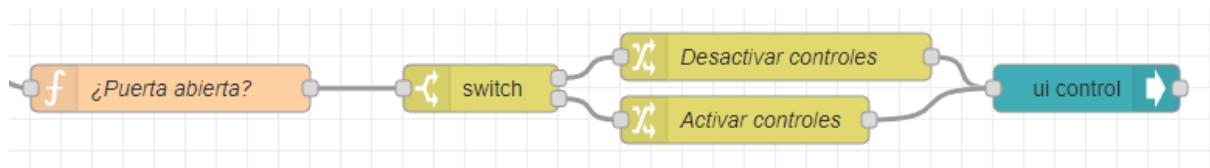
Name: ¿Puerta abierta?

Setup On Start **On Message** On Stop

```

1 if(msg.payload == 1){
2     msg.payload = "Abierta";
3 } else if(msg.payload == 0){
4     msg.payload= "Cerrada";
5 }
6
7 return msg;
  
```

Este otro bloque está diseñado para controlar la interfaz, dependiendo de que si la puerta esté cerrada o abierta, esta función hace lo mismo que la función del bloque anterior, esto lo pase a un switch el cual bifurca si el valor es true, osea si la puerta está abierta se desactivan los controles o si es false, significando que la puerta está cerrada, se activan los controles, esto pasa a un siguiente nodo llamado “ui control” que es un control de la interfaz.



Edit function node

Delete Cancel Done

Properties

Name: ¿Puerta abierta?

Setup On Start **On Message** On Stop

```

1 // Parsear el payload como un objeto JSON
2 var payloadObj = JSON.parse(msg.payload);
3
4 // Extraer la posicion del objeto
5 var posicion = payloadObj.posicion;
6
7 // Crear un nuevo mensaje con la posicion extraída
8 msg.payload = posicion;
9
10 if(msg.payload==1){
11     msg.payload = true;
12 }else if(msg.payload==0){
13     msg.payload = false;
14 }
15
16 // Devolver el mensaje modificado
17 return msg;

```

Dentro de “Desactivar controles” tenemos un código JSON para desactivar cada uno de los grupos que están ahí

Edit JSON Visual editor format JSON

```

1 {
2     "group": {
3         "hide": [
4             "Información_de_los_Sensores_Control_por_Temperatura",
5             "Información_de_los_Sensores_Switch",
6             "Información_de_los_Sensores_Tipo_de_Control"
7         ]
8     }
9 }

```

Mientras que en “Activar Controles” se hace un show de este grupo

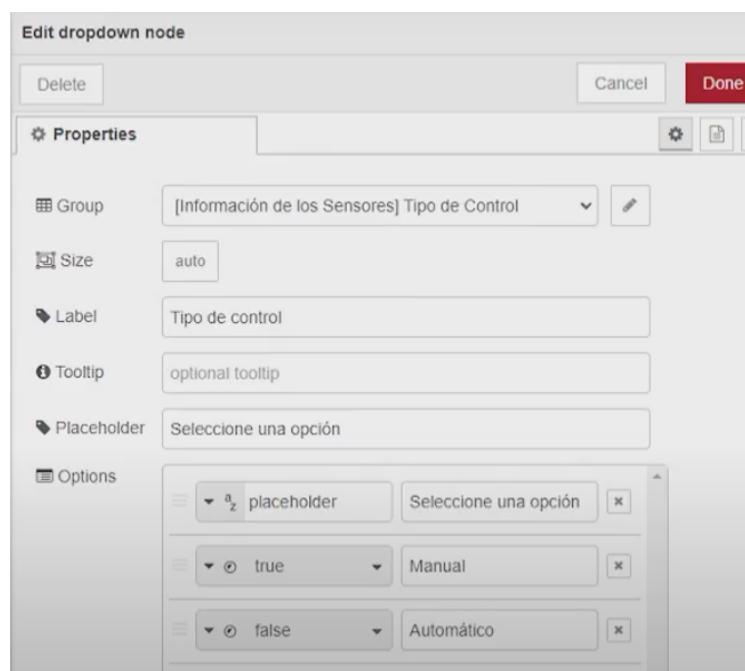
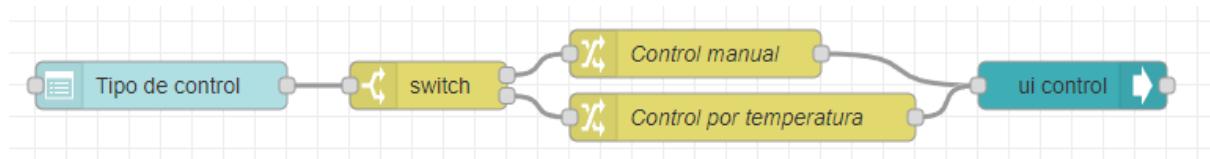
Edit JSON Visual editor format JSON

```

1  {
2    "group": {
3      "show": [
4        "Información_de_los_Sensores_Tipo_de_Control"
5      ]
6    }
7  }

```

En este otro bloque tenemos para intercambiar entre los tipos de controles en un nodo de clase dropdown, este tipo de nodo permite crear una lista desplegable en una interfaz de usuario para que el usuario pueda seleccionar una opción específica de una lista predefinida, en este nodo tenemos dos opciones, true que se le asigna el label manual y el otro que es false que se le asigna automático, esto se va a un switch y si el valor es true el control es manual, caso contrario si es false el control es por temperatura, esto finalmente pasa a un nodo “ui control” que es el control de interfaz.



Parecido al bloque anterior, dentro de “Control manual” se tiene la misma función, se muestra el grupo “información de los sensores Switch” y ocultamos “Información de los sensores de control por temperatura”.

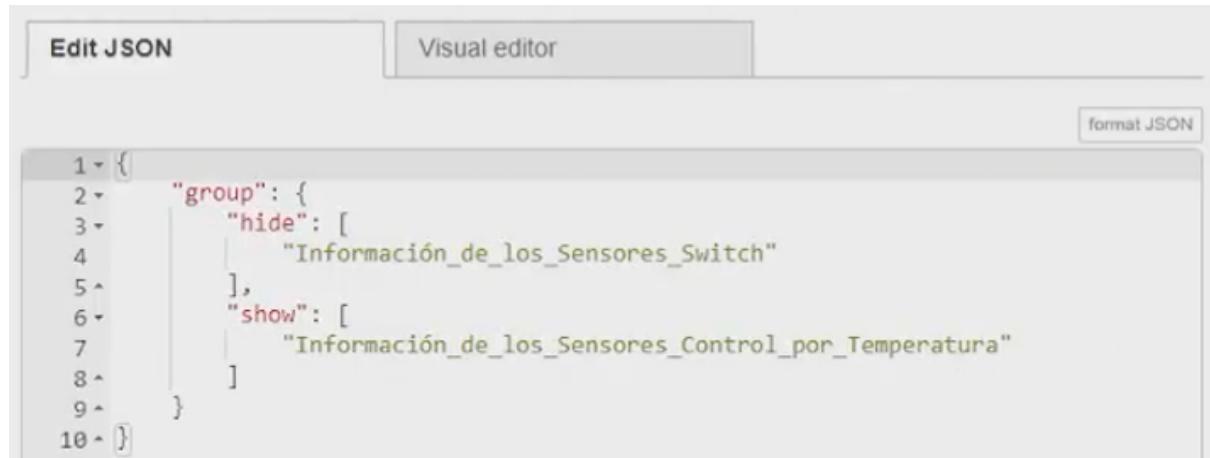


The screenshot shows the Node-RED JSON editor with the "Edit JSON" tab selected. The code in the editor is:

```
1 {  
2   "group": {  
3     "show": [  
4       "Información_de_los_Sensores_Switch"  
5     ],  
6     "hide": [  
7       "Información_de_los_Sensores_Control_por_Temperatura"  
8     ]  
9   }  
10 }
```

The "format JSON" button is visible in the top right corner.

Y dentro de “Control por temperatura” hace exactamente lo contrario.



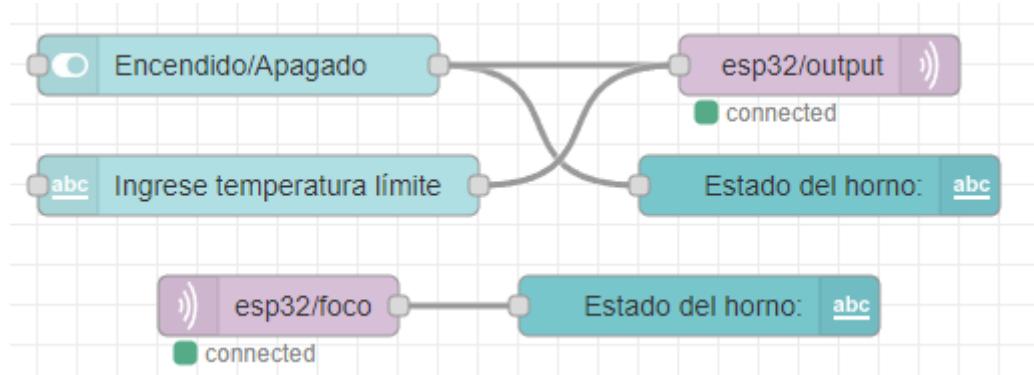
The screenshot shows the Node-RED JSON editor with the "Edit JSON" tab selected. The code in the editor is:

```
1 {  
2   "group": {  
3     "hide": [  
4       "Información_de_los_Sensores_Switch"  
5     ],  
6     "show": [  
7       "Información_de_los_Sensores_Control_por_Temperatura"  
8     ]  
9   }  
10 }
```

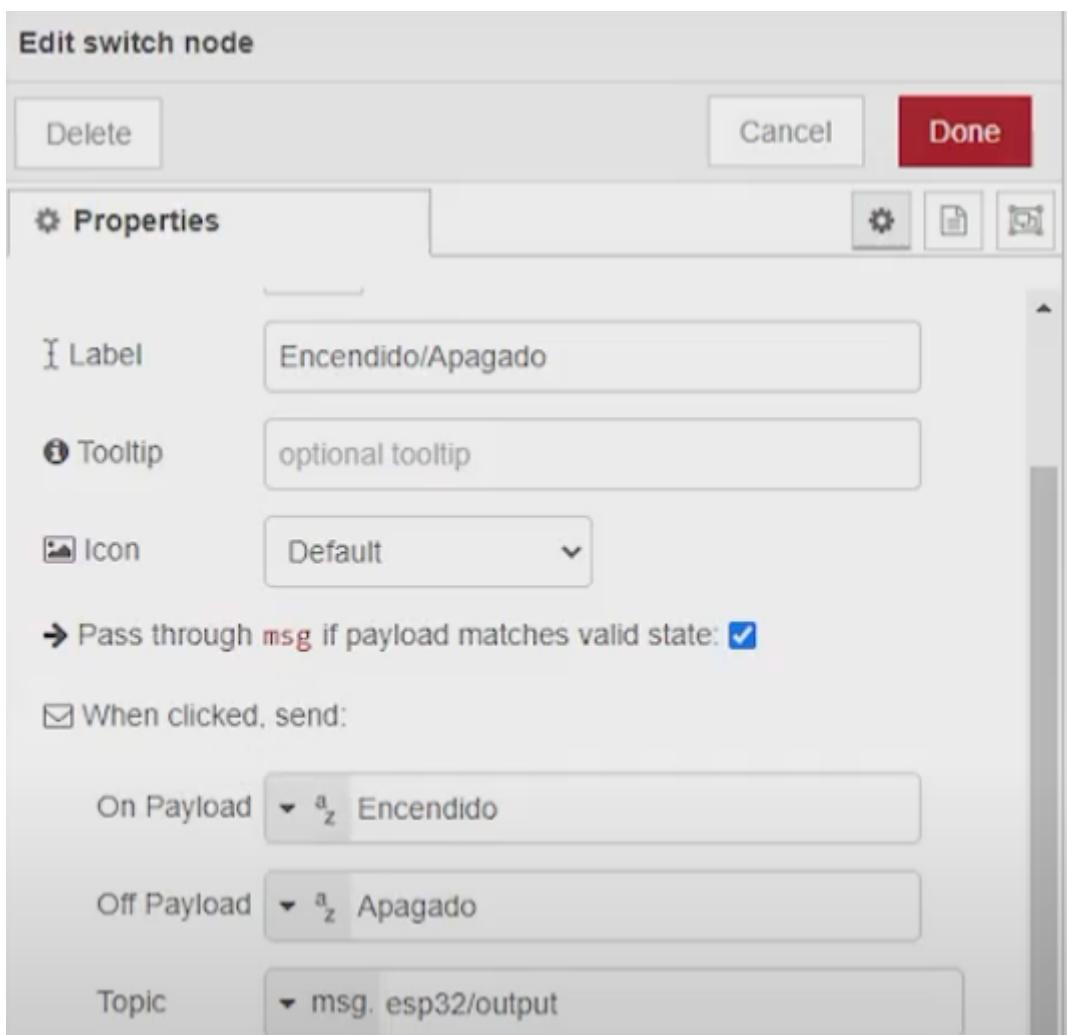
The "format JSON" button is visible in the top right corner.

Pasando al siguiente bloque, tenemos un nodo mqtt in de foco, siendo los datos del relay, que imprime en el otro nodo “Estado del horno” que nos dice si horno está encendido o apagado, luego tenemos un nodo botón de Encendido/Apagado que nos devuelve dos mensajes dependiendo de qué botón se haya pulsado, esto pasa al siguiente nodo que es un mqtt out, este es un tipo de nodo que permite enviar datos a un servidor MQTT a través de un cliente MQTT, en nuestro caso lo que hace es conectarse al broker y se envía al tópico esp32/output, a su vez, el mismo mensaje se envía a un nodo de texto que imprime el valor correspondiente al costado de la etiqueta “Estado del horno”, también tenemos otro nodo de texto que dice: “Ingrese temperatura límite”, lo introducido se envía al output.

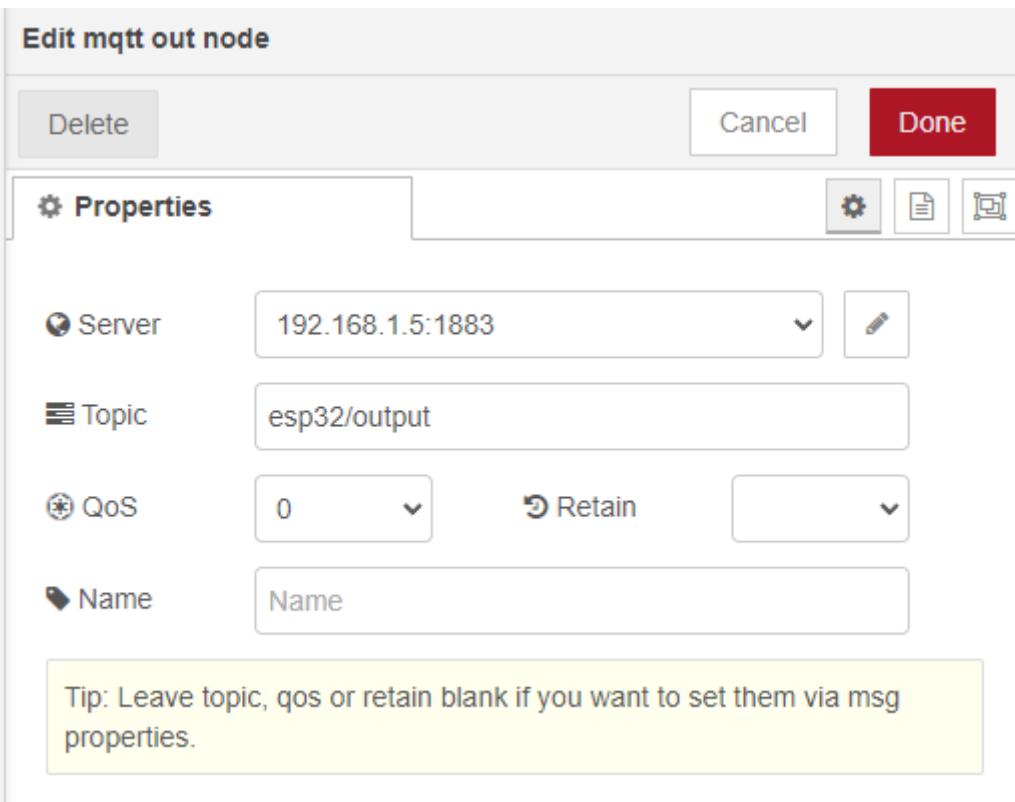
Este bloque sirve para controlar el horno ya sea de control manual (botón) o automático(temperatura límite).



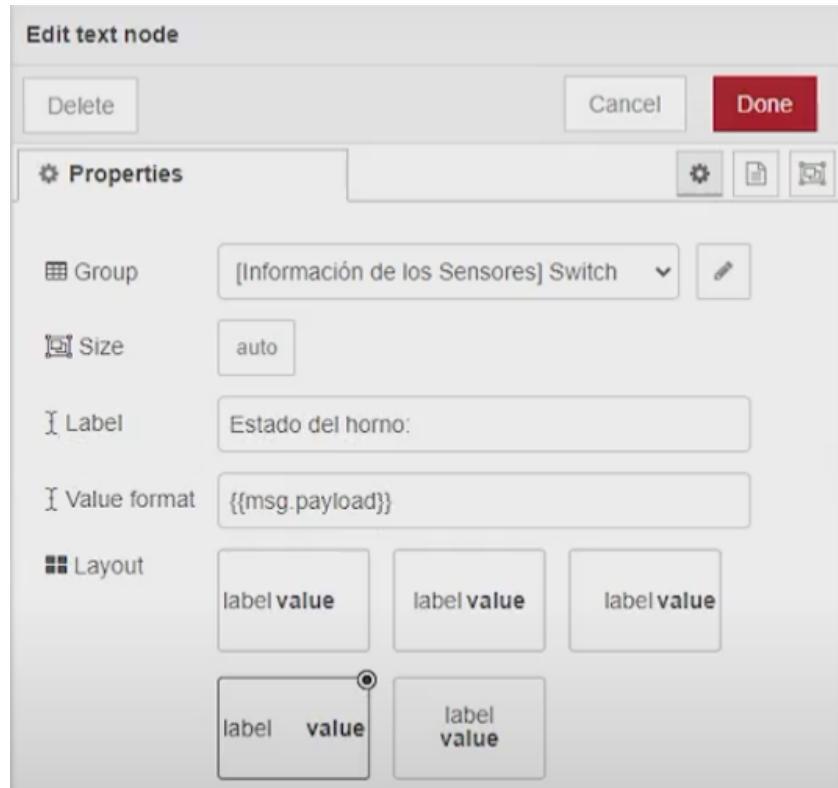
Aquí vemos dentro del nodo “Encendido/Apagado”



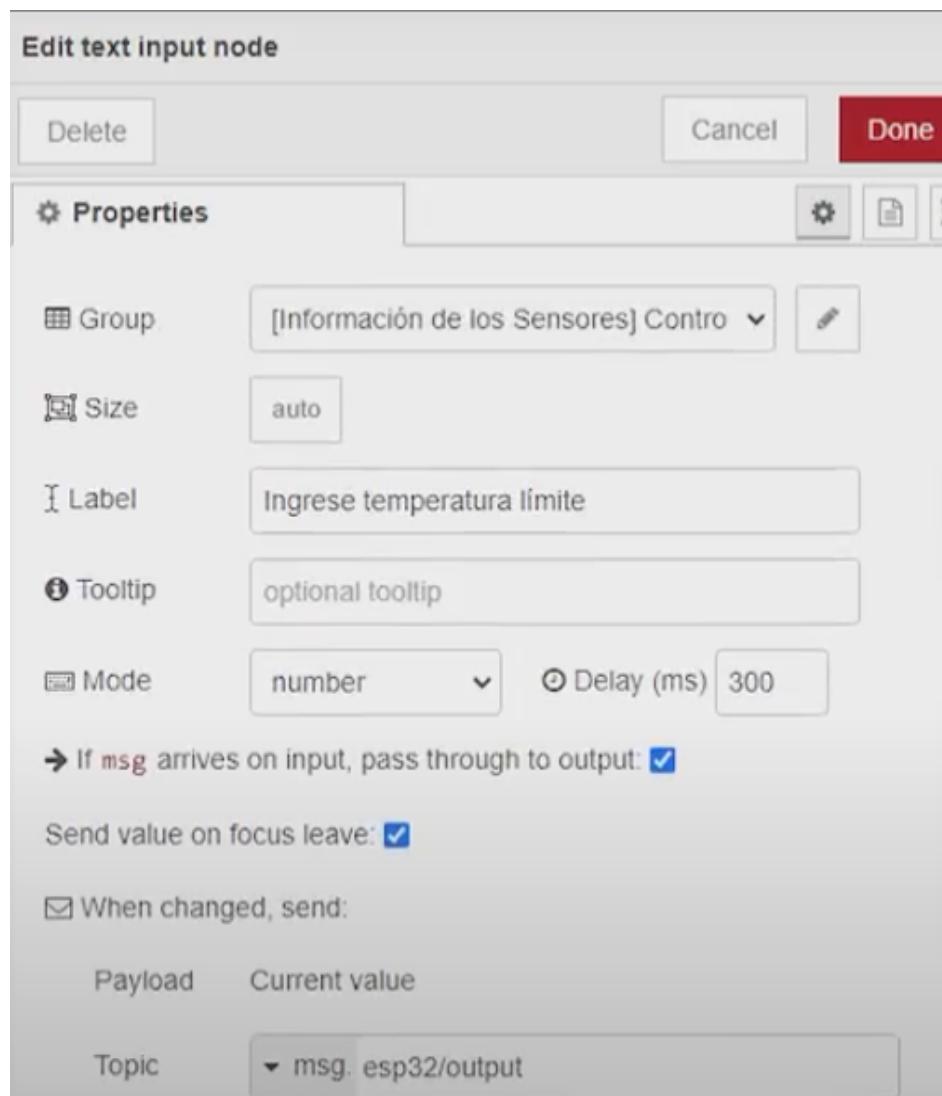
Dentro del mqtt out



Nodo de texto “Estado del horno”



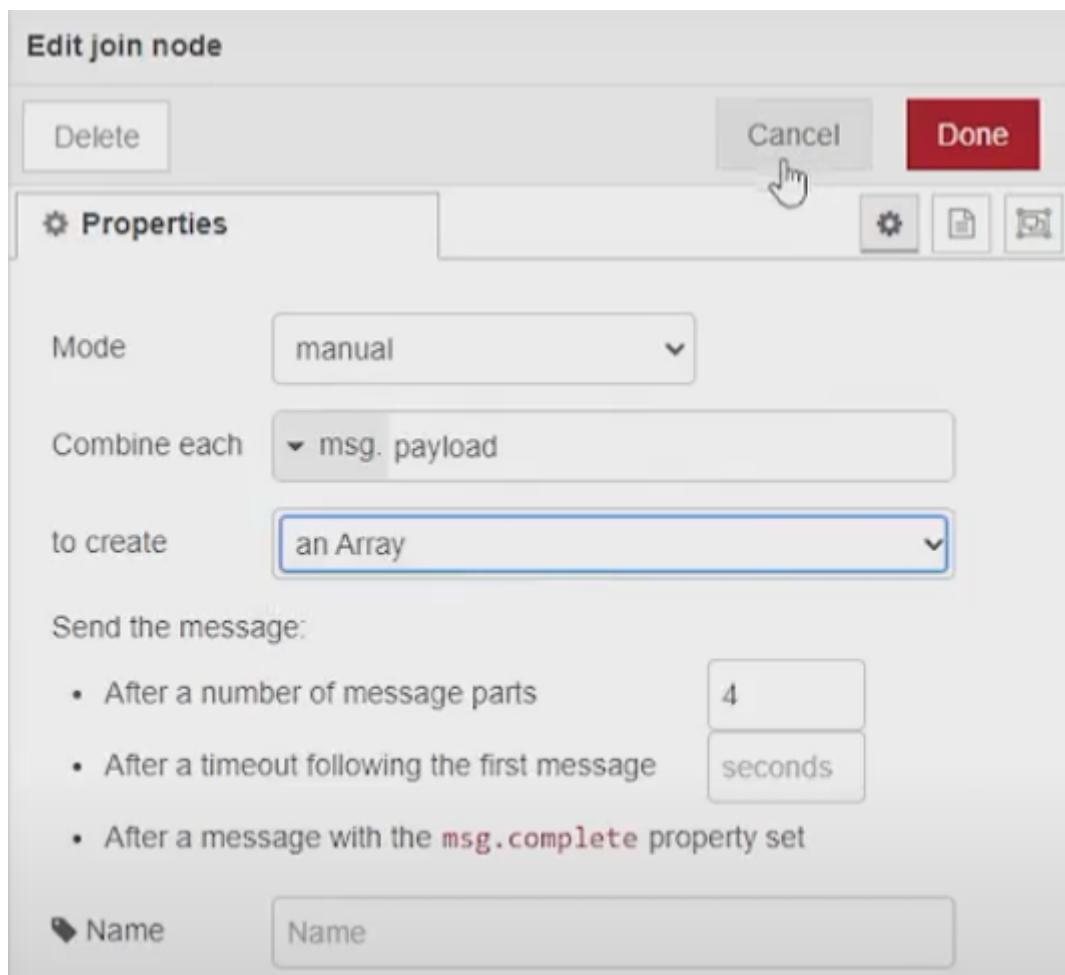
Nodo de texto “Ingrese temperatura límite”



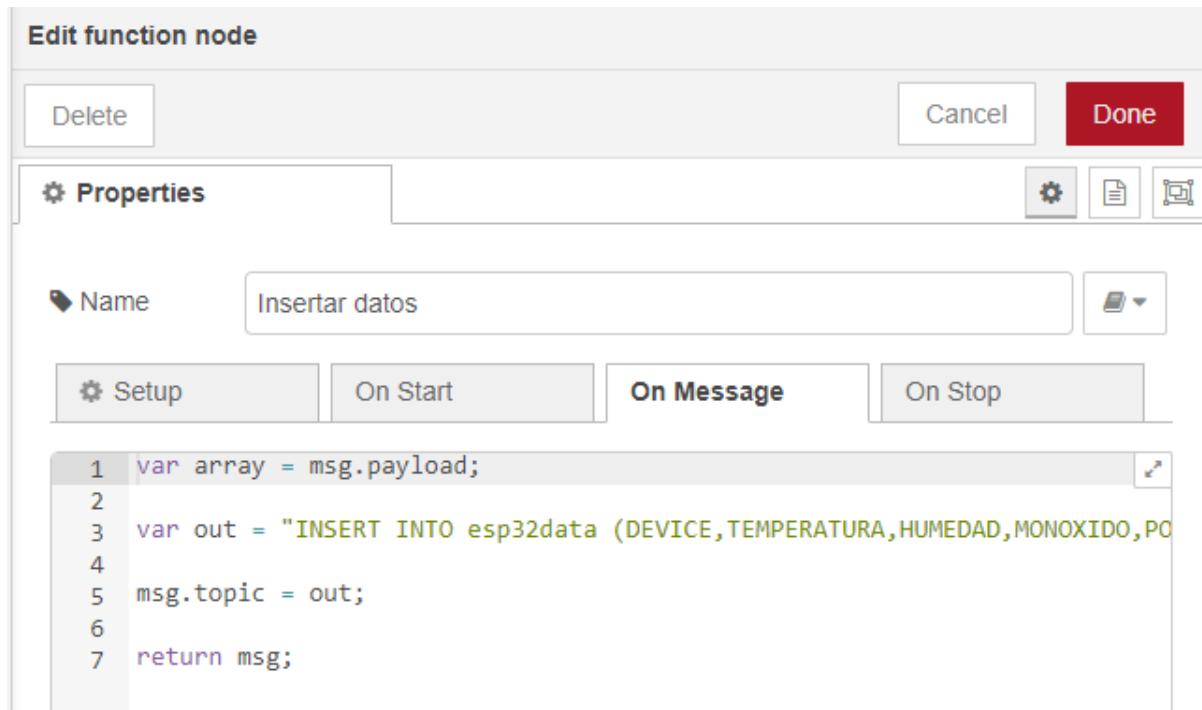
Esta parte sirve para la conexión a la base de datos mysql para almacenar los valores de los sensores, el nodo “join” lo que hace es juntar en un arreglo los cuatro valores de temperatura, humedad, posición y monóxido de carbono, y los pasa a esta función “Insertar datos” que extrae los valores del arreglo e insertarlos en la tabla, esto finalmente se transmite al nodo “db_sensores” que es donde finalmente se ejecuta en la base de datos, en el host tenemos la ip de localhost, el puerto por defecto de Xampp, el usuario root y la database llamada “db_sensores”



Nodo “join” donde se escoge array



Nodo función “Insertar datos” donde está el código que inserta los datos en la tabla “esp32data”



Nodo mysql

Edit mysql node > **Edit MySQL database node**

[Delete](#) [Cancel](#) [Update](#)

Properties

Host	192.168.1.5
Port	3306
User	root
Password	(redacted)
Database	db_sensores
Timezone	±hh:mm
Charset	UTF8
Name	Name

Aca tenemos las credenciales del Emqx, vemos las conexiones, los tópicos conectados y las suscripciones

EMQX

Dashboard

[Overview](#) [Nodes](#) [Metrics](#)

[Upgrade →](#)

Incoming Rate: 4 messages/sec

Outgoing Rate: 4 messages/sec

Connections: 53 **Topics**: 6 **Subscriptions**: 6

Node Data

Node Name: emqx@172.18.0.3 Node Role: core
Uptime: 1 hour 24 minutes 39 seconds Version: 5.0.16
Connections: 53 File Descriptors Limit: 1048576
Subscriptions: 6 OS CPU Load: 0.1/0.07/0.08
Topics: 6 OS Memory:

Dashboard **Nodes** **Metrics**

Connections **Subscriptions** **Retained Messages** **Access Control** **Data Integration** **Configuration** **Extensions** **Diagnose**

En esta sección ““Subscription” se realiza la suscripción a los temas (topics) de interés para recibir mensajes publicados por los dispositivos conectados al broker MQTT.

Subscription

Topic	QoS	Actions
esp32/co	0	<button>Subscribe</button>
esp32/co	0	<button>Cancel</button>
esp32/posicion	0	<button>Cancel</button>
esp32/humedad	0	<button>Cancel</button>
esp32/temperatura	0	<button>Cancel</button>

En esta sección ““Publish” se realiza el envío de mensajes desde los dispositivos conectados al broker MQTT a los suscriptores interesados en recibirllos.

Publish

Topic	Payload	QoS	Actions
testtopic/1	{"msg": "hello"}	0	<input type="checkbox"/> Retain <button>Publish</button>

Received

Topic	QoS	Payload	Time
esp32/co	0	{"monóxido de carbón": 1}	2023-03-09 01:19:20
esp32/posicion	0	{"posicion": 1}	2023-03-09 01:19:20
esp32/humedad	0	{"humedad": 59.00}	2023-03-09 01:19:20
esp32/temperatura	0	{"temperatura": 28.5}	2023-03-09 01:19:20

Published

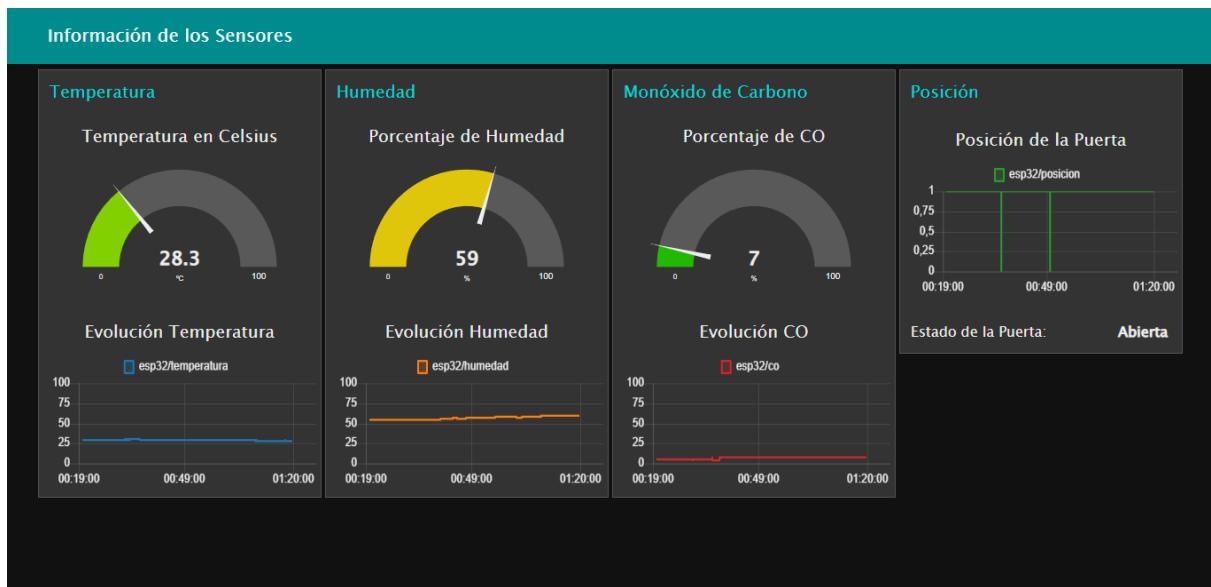
Topic	QoS	Payload	Time
No Data			

5. Resultados

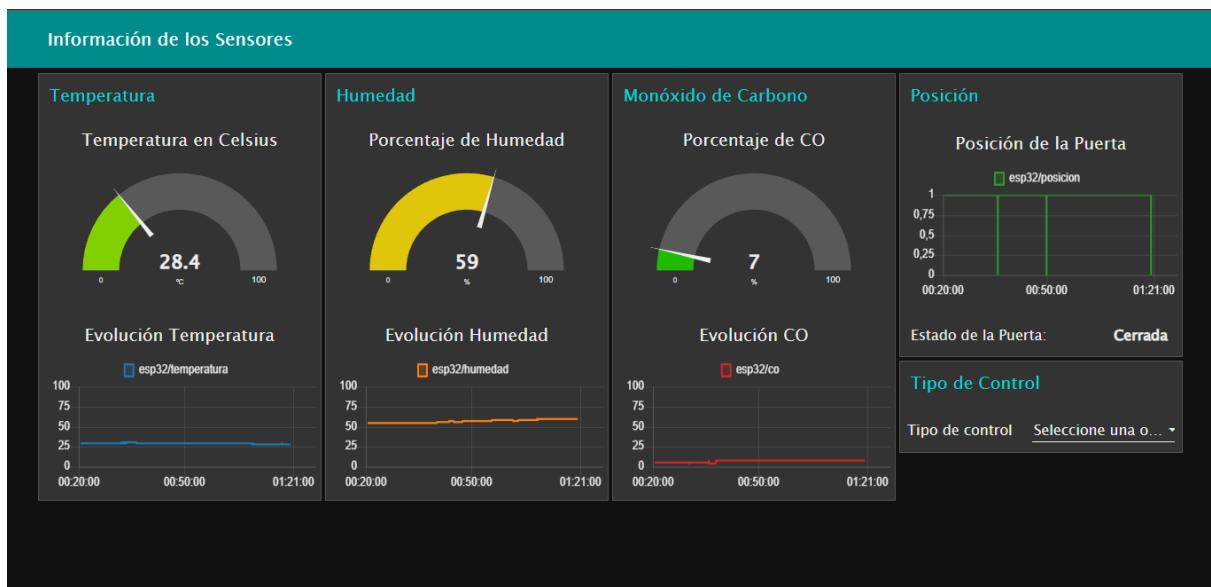
Obtuvimos los siguientes resultados:

En las siguientes imágenes podremos ver el diseño del dashboard y estos 4 serán distintos debido a su situación.

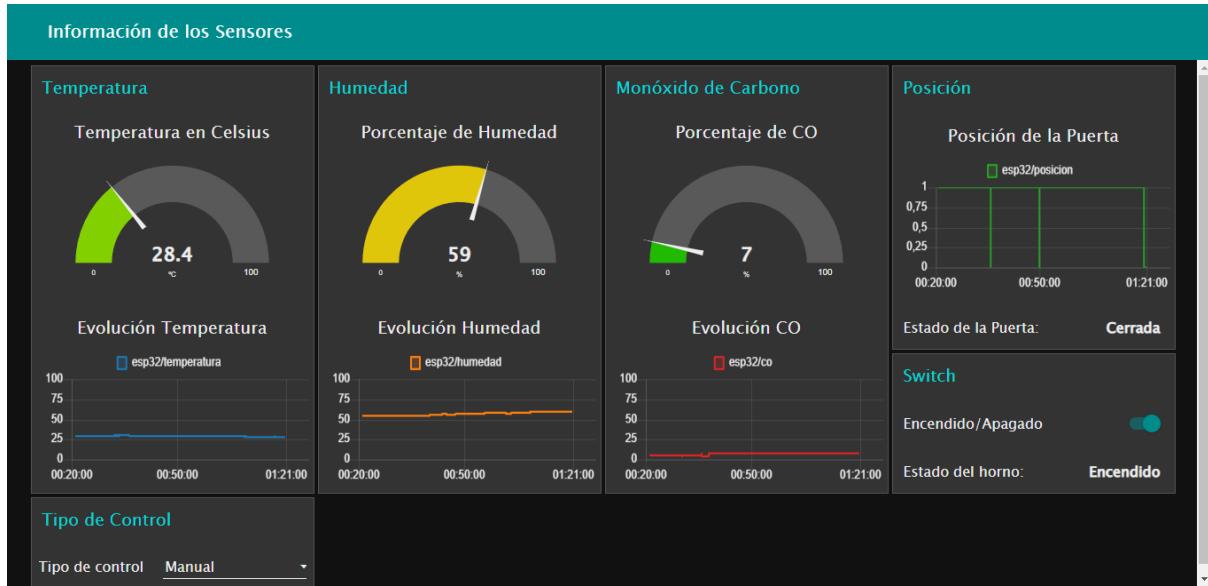
En la siguiente imagen no existe algún bloque que nos permita controlarlo sea de forma automática o manual debido a que la puerta no se encuentra cerrada.



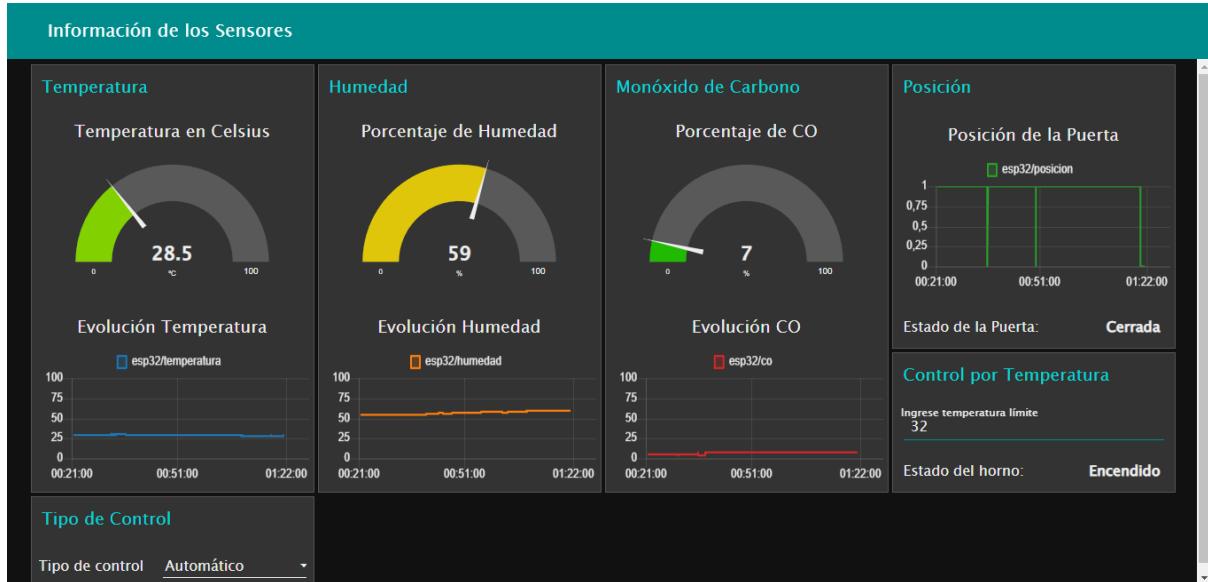
Una vez que la puerta se encuentre cerrada, podremos observar que aparece un nuevo bloque que nos da a escoger el tipo de control: automático o manual.



En tipo de Control se puede observar que se encuentra en manual y aparece el bloque de switch que nos permite encenderlo y apagarlo de forma manual.



A diferencia de la anterior imagen, esta vez, se encuentra en control automático. En control por temperatura debemos colocar el número que representará el límite. Cuando se llegue a este límite se apagará el horno.



6. Conclusiones

- El sistema de control on-off y monitoreo usando el ESP32 permitió mantener la temperatura del horno dentro de un rango deseado, evitando que se produzcan temperaturas demasiado altas. Además, el monitoreo de la temperatura permite una supervisión constante del proceso de calentamiento, lo que puede mejorar la eficiencia y la calidad del producto final. En resumen, este sistema puede ser una herramienta útil para aplicaciones industriales o de laboratorio que requieran un control de la temperatura.
- El uso del broker EMQX, la base de datos MySQL y el dashboard desarrollado en Node-RED permiten a los usuarios controlar y monitorear el sistema de manera efectiva. El broker MQTT en EMQX actúa como intermediario entre los dispositivos y la base de datos, permitiendo una comunicación bidireccional segura y confiable. La base de datos MySQL almacena los datos en tiempo real generados por los dispositivos y los usuarios pueden acceder a los datos de manera fácil y rápida. Por último, el dashboard desarrollado en Node-RED proporciona una interfaz gráfica intuitiva que permite a los usuarios controlar el sistema y visualizar los datos en tiempo real. En resumen, la implementación de estos servicios es esencial para el éxito de la aplicación y su uso puede mejorar la eficiencia, seguridad y confiabilidad del sistema.
- El dashboard implementado en node-red permite visualizar el estado del horno así como la temperatura, humedad, posición y monóxido, que son críticas para el funcionamiento seguro y eficiente del horno. Esto permite a los usuarios tomar decisiones informadas y rápidas en caso de una situación de emergencia.

7. Bibliografía

Rascon Madrigal, L. H. (2022). Registro automático de temperatura y activación de alarmas en cuatro zonas de un horno de extrusión. *Instituto de Ingeniería y Tecnología*. Recuperado de <http://cathi.uacj.mx/20.500.11961/24966>

Bravo Cedeño, G. S. (2020). *Diseño e implementación de prototipo de sistema electrónico monitoreado en tiempo real para programa de producción de fábrica de vidrio* (Doctoral dissertation, Universidad de Guayaquil. Facultad de Ciencias Matemáticas y Físicas. Carrera

de Ingeniería en Networking y Telecomunicaciones). Recuperado de
<http://repositorio.ug.edu.ec/handle/redug/49903>

Paspuel Pozo, T. L. (2022). *Sistema inteligente de alerta de incendios o fugas de gas licuado de petróleo para viviendas de adultos mayores utilizando el protocolo Lora* (Bachelor's thesis). Recuperado de <http://repositorio.utn.edu.ec/handle/123456789/12613>

Espinoza Dueñas, B. (2019). Sistema automatizado para la identificación de defectos y la clasificación de baldosas cerámicas en la salida del horno. Recuperado de
<https://hdl.handle.net/20.500.12867/2525>

Espressif Systems. (2021). ESP32. Recuperado el 9 de marzo de 2023, de
<https://www.espressif.com/en/products/socs/esp32/overview>

Random Nerd Tutorials. (2021). Getting Started with ESP32. Recuperado el 9 de marzo de 2023, de <https://randomnerdtutorials.com/getting-started-with-esp32/>

8. Anexos

8.1. Anexo 1: Código docker para despliegue de servicio EMQX y node-red version: "3.8"

El primer servicio, emqx, utiliza la imagen "emqx/emqx:5.0.16" para crear un contenedor con el nombre "emqx". El servicio se configura para reiniciarse siempre que se detenga. Además, se exponen tres puertos del contenedor al host: 1883, 8083 y 18083.

El segundo servicio, nodered, utiliza la imagen "nodered/node-red:2.2.3" para crear un contenedor con el nombre "node-red". El servicio se configura para reiniciarse siempre que se detenga. Además, se expone el puerto 1880 del contenedor al host. También se establece un volumen de datos en el host, mapeando el directorio local "./nodered/" al directorio "/data" en el contenedor. Esto permite que los flujos y la configuración de Node-RED se guarden en el host, lo que hace que el contenedor sea más portátil y más fácil de gestionar.

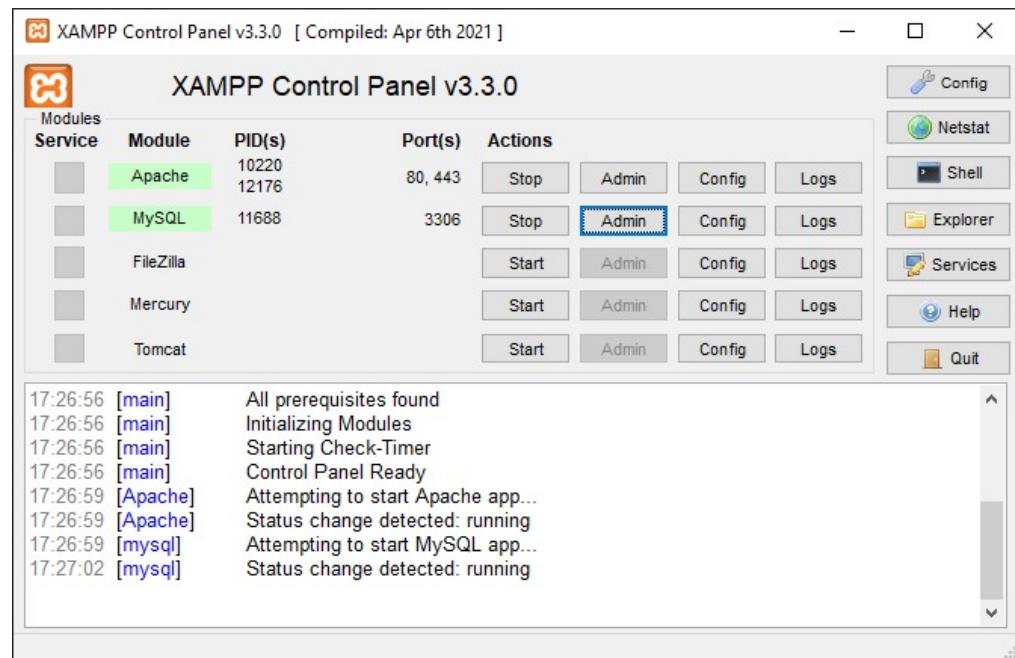
```
services:  
  # Configuración del servicio EMQX  
  emqx:  
    container_name: emqx  
    image: emqx/emqx:5.0.16  
    restart: always  
    # Mapeo de puertos para permitir la comunicación con el host
```

ports:
- 1883:1883
- 8083:8083
- 18083:18083

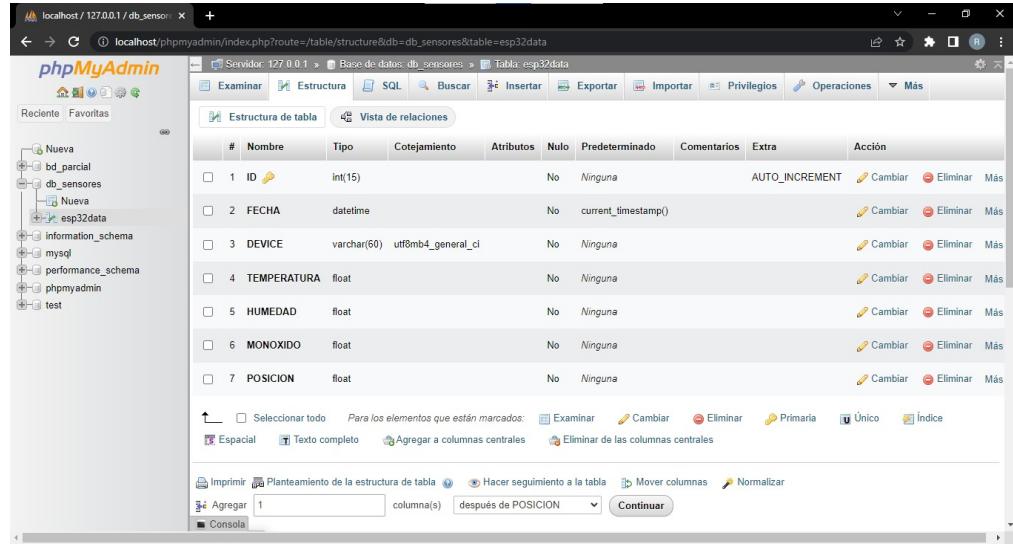
```
# Configuración del servicio Node-RED
nodered:
  container_name: node-red
  image: nodered/node-red:2.2.3
  restart: always
  # Mapeo de puerto para permitir la comunicación con el host
  ports:
    - 1880:1880
  # Montaje de un volumen para permitir la persistencia de los flujos y la
  # configuración de Node-RED
  volumes:
    - ./nodered/:/data
```

8.2. Anexo 2: Despliegue y configuración de la base de datos.

Para el despliegue de la base de datos se usó xampp, ya que incluye una interfaz visual de diseño para la base de datos.



Estructura de la tabla esp32data que es en donde se almacenará el valor de las medidas de los sensores.



8.3. Anexo 3: Código en arduino para la programación del ESP32

```

#include <LiquidCrystal_I2C.h> // Librería para el LCD
#include <WiFi.h>           // Librería para el WiFi del ESP32
#include <PubSubClient.h>     // Librería para conexión con el servidor MQTT
#include <DHTesp.h>          // Librería para el DHT11

// CONSTANTES
#define DHT 4    // Se define el pin del sensor de temperatura y humedad
#define MQ9 32   // Se define el pin del sensor de CO2
#define POS 19   // Se define el pin del sensor de posición
#define RELAY 2  // Se define el pin del relay

// VARIABLES
float temperatura = 0; // Se inicializa la variable de la temperatura
float humedad = 0;      // Se inicializa la variable de la humedad
bool cerrado = false;   // Se inicializa la variable de la posición de la puerta
int co = 0;              // Se inicializa la variable de la concentración de CO2
float setpoint = 0;      // Se inicializa el umbral de control por temperatura
const float band = 0;    // Se define la banda de tolerancia del control por
                        // temperatura

// Se inicializa el dht
DHTesp dht; // Se crea un objeto del tipo DHTesp para manejar el sensor DHT11

// Se crea el objeto LCD con dirección 0x3F y 16 columnas x 2 filas
LiquidCrystal_I2C lcd(0x27, 16, 2); //

// Configuración de la red WiFi
const char* ssid = "TP-LINK_7D36DE";
const char* password = "30744738";

```

```

// Configuración del servidor EMQ X
const char* mqtt_server = "192.168.1.5";
const int mqtt_port = 1883;
const char* mqtt_user = "admin";
const char* mqtt_password = "admin123";

// Configuración del cliente MQTT
WiFiClient espClient;
PubSubClient client(espClient);

void setup() {

    // Inicialización del puerto serie
    Serial.begin(115200);      // Se inicia la comunicación serial
    dht.setup(DHT, DHTesp::DHT11); // Se inicializa el sensor de temperatura y
    humedad
    pinMode(POS, INPUT);       // Se configura el pin del sensor de posición como
    entrada
    pinMode(RELAY, OUTPUT);
    Serial.println("Configuracion Inicial Realizada ...."); // Se envía un mensaje por
    la comunicación serial indicando que se ha terminado la configuración inicial

    // Conexión a la red WiFi
    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED) {
        delay(1000);
        Serial.println("Conectando a la red WiFi...");
        Serial.println(WiFi.status());
    }
    Serial.println("Conexión a la red WiFi establecida");

    // Conexión al servidor MQTT EMQ X
    client.setServer(mqtt_server, mqtt_port);
    client.setCallback(callback);

    while (!client.connected()) {
        Serial.println("Conectando al servidor MQTT...");
        if (client.connect("ESP32", "admin", "admin123")) {
            Serial.println("Conectado al servidor MQTT");
            client.subscribe("esp32/output");
            client.publish("esp32/output", "Mensaje de prueba");
        } else {
            Serial.print("Falló la conexión al servidor MQTT, rc=");
            Serial.println(client.state());
            delay(5000);
        }
    }
}

```

```

// Se inicializa el LCD
lcd.begin();

// Se enciende la luz del LCD
lcd.backlight();
}

void loop() {

    // Si no existe conexión con el servidor MQTT se reintenta la conexión
    if (!client.connected()) {
        reconnect();
    }
    client.loop();

    // Toma de lecturas
    TempAndHumidity data = dht.getTempAndHumidity(); // Se lee la temperatura
    y humedad del sensor DHT11
    temperatura = data.temperature;           // Se actualiza la variable de la
    temperatura
    humedad = data.humidity;                  // Se actualiza la variable de la
    humedad
    cerrado = digitalRead(POS);             // Se lee la posición de la puerta
    co = map(analogRead(MQ9), 0, 4095, 0, 100); // Se lee la concentración de
    CO2 y se mapea a un rango de 0 a 100

    //Salida Serial
    Serial.print("Temperatura: "); // Se envía un mensaje por la comunicación serial
    indicando que se va a imprimir la temperatura
    Serial.println(temperatura); // Se imprime la temperatura por la comunicación
    serial
    Serial.print("Humedad: "); // Se envía un mensaje por la comunicación serial
    indicando que se va a imprimir la humedad
    Serial.println(humedad); // Se imprime la humedad por la comunicación
    serial
    Serial.print("Posicion: "); // Se envía un mensaje por la comunicación serial
    indicando que se va a imprimir la posición de la puerta
    Serial.println(cerrado); // Se imprime la posición de la puerta por la
    comunicación serial
    Serial.print("CO: "); // Se envía un mensaje por la comunicación serial
    indicando que se va a imprimir la concentración de CO2
    Serial.println(co); // Se imprime la concentración de CO2 por la
    comunicación serial

    // Publicación de los datos en el servidor MQTT
    if (!isnan(temperatura)) {

```

```

String message = "{\"temperatura\":\"" + String(temperatura) + "\"}; // Se crea un
String en formato JSON que contiene los datos que deseamos enviar
client.publish("esp32/temperatura", message.c_str()); // Se publica el
mensaje en el servidor MQTT dentro del tópico esp32/temperatura
Serial.println("Datos de temperatura publicados en el servidor MQTT");
} else {
    Serial.println("Error al leer los datos de temperatura");
}

if (!isnan(humedad)) {
    String message = "{\"humedad\":\"" + String(humedad) + "\"";
    client.publish("esp32/humedad", message.c_str());
    Serial.println("Datos de humedad publicados en el servidor MQTT");
} else {
    Serial.println("Error al leer los datos de humedad");
}

if (!isnan(cerrado)) {
    String message = "{\"posicion\":\"" + String(cerrado) + "\"";
    client.publish("esp32/posicion", message.c_str());
    Serial.println("Datos de posicion publicados en el servidor MQTT");
} else {
    Serial.println("Error al leer los datos de posicion");
}

if (!isnan(co)) {
    String message = "{\"monóxido de carbono\":\"" + String(co) + "\"";
    client.publish("esp32/co", message.c_str());
    Serial.println("Datos de co publicados en el servidor MQTT");
} else {
    Serial.println("Error al leer los datos de co");
}

// Imprimir los valores de temperatura, humedad, cerrado y co en la pantalla
LCD
lcd.setCursor(0, 0);
lcd.print("Temp: ");
lcd.print(temperatura);
lcd.print("C ");

lcd.setCursor(0, 1);
lcd.print("Hum: ");
lcd.print(humedad);
lcd.print("% ");

lcd.setCursor(14, 0);
lcd.print("Pos: ");

```

```

lcd.print(cerrado);

lcd.setCursor(13, 1);
lcd.print("CO: ");
lcd.print(co);
lcd.print("%");

// Se realiza un autoscroll para mostrar todos los datos en el LCD
lcd.autoscroll();

// Espera de 1 segundo
delay(1000);
}

// Esta función es el callback que se ejecuta cuando llega un mensaje al cliente
MQTT
void callback(char* topic, byte* payload, unsigned int length) {
    // Imprime el mensaje recibido y el tema al que pertenece
    Serial.print("Mensaje recibido [");
    Serial.print(topic);
    Serial.print("] ");

    // Crea una variable String para almacenar el mensaje recibido
    String messageTemp;

    // Recorre los bytes del mensaje recibido y los imprime en el monitor serial
    for (int i = 0; i < length; i++) {
        Serial.print((char)payload[i]);
        // Agrega cada byte del mensaje a la variable messageTemp
        messageTemp += (char)payload[i];
    }

    // Verifica si el tema del mensaje recibido es "esp32/output"
    if (String(topic) == "esp32/output") {
        // Si el mensaje es "Encendido", activa el relé y publica un mensaje en el tema
        "esp32/foco"
        if (messageTemp == "Encendido") {
            digitalWrite(RELAY, HIGH);
            client.publish("esp32/foco", "Encendido");
        }
        // Si el mensaje es "Apagado", desactiva el relé y publica un mensaje en el tema
        "esp32/foco"
        else if (messageTemp == "Apagado") {
            digitalWrite(RELAY, LOW);
            client.publish("esp32/foco", "Apagado");
        }
    }
}

```

```

// Si el mensaje no es "Encendido" ni "Apagado", se asume que es el valor
deseado del setpoint para controlar la temperatura
else {
    // Imprime el valor deseado del setpoint en el monitor serial
    Serial.print("Cambiando el setpoint a ");
    Serial.println(messageTemp);
    // Convierte el valor deseado del setpoint a un entero y lo guarda en la
    variable setpoint
    setpoint = messageTemp.toInt();
    // Imprime el valor actual de setpoint en el monitor serial
    Serial.println(setpoint);
    // Si la temperatura actual es mayor que el setpoint más la banda de
    tolerancia, se apaga el relé y se publica un mensaje en el tema "esp32/foco"
    if (temperatura > setpoint + band) {
        digitalWrite(RELAY, LOW);
        client.publish("esp32/foco", "Apagado");
    }
    // Si la temperatura actual es menor que el setpoint menos la banda de
    tolerancia, se enciende el relé y se publica un mensaje en el tema "esp32/foco"
    else if (temperatura < setpoint - band) {
        digitalWrite(RELAY, HIGH);
        client.publish("esp32/foco", "Encendido");
    }
}
}

void reconnect() {
    // Loop que se ejecuta hasta que se haya reconectado al servidor MQTT
    while (!client.connected()) {
        Serial.println("Conectando al servidor MQTT...");
        if (client.connect("ESP32", "admin", "admin123")) {
            Serial.println("Conectado al servidor MQTT");
            client.subscribe("esp32/output");
            client.publish("esp32/output", "Mensaje de prueba");
        } else {
            Serial.print("Falló la conexión al servidor MQTT, rc=");
            Serial.println(client.state());
            delay(5000);
        }
    }
}

```