



UNIVERSIDAD NACIONAL DE INGENIERÍA
FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA
PROGRAMACION ORIENTADA A OBJETOS (BMA15)



GRUPO 4º: DOCUMENTACION DEL PROYECTO

Integrantes

Nº	Código	Apellidos y nombres
1	:20231452I	: Herrera Valencia Nancy Alexandra
2	:20232693J	: Maverick Wenceslao Santos del Pino

Sección: N

Docente: Tello Canchapoma, Yuri Oscar

2024-III

Contenido

Introducción.....	3
Objetivos Generales.....	4
Objetivos Específicos.....	4
Las Cuatro Revoluciones Industriales.....	6
Industria 4.0.....	7
IOT.....	10
MQTT.....	14
Cliente MQTT.....	15
Broker MQTT.....	15
Topic MQTT.....	15
Circuito Simulado.....	17
Circuito Electrónico.....	20
Chip ESP32.....	22
Sensor DHT11.....	23
Senamhi.....	24
Protoboard.....	25
Resistencia.....	26
Jumper.....	27
Presupuesto.....	28
Proyectos similares.....	28
Tecnologías utilizadas.....	32
UML.....	34
Diagrama de bloques.....	39
Código Arduino.....	41
Código Python.....	46
Conclusiones.....	56
Referencias.....	58

Introducción

El proyecto ShieldNet toma su nombre de la combinación de las palabras en inglés "shield" (escudo) y "net" (red), reflejando su propósito de proteger y supervisar redes IoT. Es un Sistema Inteligente de Detección y Protección en Redes IoT que no solo se encarga de monitorear variables físicas como la temperatura, sino que también analiza el tráfico de datos en la red para identificar patrones sospechosos. Este análisis permite detectar comportamientos anómalos, como accesos no autorizados o irregularidades en las comunicaciones, proporcionando una capa adicional de seguridad a los dispositivos conectados.

En el contexto de la Internet de las Cosas (IoT), la monitorización remota de variables ambientales es esencial para aplicaciones como la automatización industrial, el control de climatización y la detección de fallos en tiempo real. Este proyecto implementa un sistema que detecta anomalías en la temperatura utilizando un ESP32 conectado a un sensor DHT11. Los datos recopilados se publican a través del protocolo MQTT hacia un broker, permitiendo que otros dispositivos o programas los consuman en tiempo real. Este enfoque facilita la creación de un sistema distribuido, eficiente y adaptable a diversas aplicaciones críticas.

El ESP32 es un microcontrolador de bajo costo y alto rendimiento con capacidades de conectividad Wi-Fi y Bluetooth, ideal para proyectos de IoT. En este sistema, el ESP32 se encarga de leer las mediciones del sensor DHT11, un sensor digital ampliamente utilizado para registrar temperatura y humedad debido a su simplicidad y precisión aceptable en aplicaciones no críticas. Las lecturas de temperatura se envían periódicamente a un broker MQTT, un intermediario que permite la comunicación asíncrona entre dispositivos mediante un modelo publicador-suscriptor, optimizando así la transferencia de datos en entornos distribuidos.

En el lado del software, se utiliza Python para suscribirse a los tópicos del broker MQTT, recibir

los datos de temperatura y analizarlos en tiempo real. Este análisis puede incluir la identificación de anomalías mediante algoritmos estadísticos o de aprendizaje automático. Además, se visualizan las temperaturas en una gráfica dinámica, lo que permite un monitoreo intuitivo y en tiempo real de las variaciones térmicas. Este enfoque integral no solo permite detectar desviaciones anómalas que podrían indicar fallos en el sistema, sino que también establece una base escalable para la incorporación de sensores adicionales o el ajuste de parámetros de alerta.

Objetivos Generales

Diseñar e implementar una plataforma educativa que permita el análisis y monitoreo de redes IoT utilizando datos reales, con el propósito de identificar vulnerabilidades, patrones de tráfico y comportamientos anómalos, promoviendo el aprendizaje práctico y el fortalecimiento de la seguridad en dispositivos conectados a través de herramientas accesibles y visualizaciones claras.

Objetivos Específicos

1. Implementación de un sistema de monitoreo seguro:

- Diseñar y configurar un sistema de monitoreo para redes IoT que priorice la seguridad, enfocándose en la detección de patrones de tráfico no autorizados o inconsistentes en dispositivos conectados.

2. Identificación de vulnerabilidades en dispositivos IoT:

- Desarrollar herramientas para evaluar y clasificar vulnerabilidades en los dispositivos IoT conectados, considerando configuraciones inadecuadas, puertos expuestos y autenticaciones débiles.

3. Integración de técnicas ligeras de detección de anomalías:

- Aplicar algoritmos de bajo costo computacional para detectar comportamientos anómalos en redes IoT que puedan sugerir intentos de intrusión o fallos operativos, sin requerir recursos avanzados de hardware.

4. Análisis del impacto de los datos reales en redes IoT:

- Realizar un análisis del tráfico real de datos en redes IoT para identificar tendencias y patrones de uso, con el objetivo de optimizar la configuración y mejorar la eficiencia de los dispositivos conectados.

5. Educación y capacitación en el uso de herramientas de monitoreo:

- Desarrollar un módulo educativo práctico que permita a los usuarios comprender e interpretar los datos recopilados, promoviendo una mejor toma de decisiones en la gestión de redes IoT.

6. Documentación y análisis del proceso:

- Generar un reporte detallado que documente el funcionamiento, limitaciones y oportunidades de mejora del sistema diseñado, resaltando las lecciones aprendidas en el análisis de redes IoT reales.

Las Cuatro Revoluciones Industriales

Primera Revolución Industrial

A finales del siglo XVIII en Gran Bretaña, surgió la primera revolución industrial, marcada por el uso de energía hidráulica y de vapor en lugar de la fuerza manual o animal. Este avance permitió la producción en masa, ya que las máquinas comenzaron a fabricar productos terminados, reemplazando el trabajo artesanal detallado.(IBM, s.f)

Segunda Revolución Industrial

A finales del siglo XIX, la segunda revolución industrial introdujo las líneas de ensamblaje y nuevas fuentes de energía como el petróleo, el gas y la electricidad. Además, los avances en las comunicaciones, como el teléfono y el telégrafo, facilitaron la automatización parcial y aumentaron la capacidad de producción a gran escala. (IBM, s.f)

Tercera Revolución Industrial

Iniciada a mediados del siglo XX, la tercera revolución industrial trajo consigo la incorporación de ordenadores, telecomunicaciones avanzadas y el análisis de datos en los procesos productivos. La introducción de controladores lógicos programables (PLC) permitió automatizar tareas, recopilar información y compartirla para mejorar la eficiencia operativa. (IBM, s.f)

Cuarta Revolución Industrial

Actualmente vivimos en la cuarta revolución industrial, conocida como Industria 4.0, caracterizada por un alto grado de automatización y el desarrollo de fábricas inteligentes. La integración de grandes volúmenes de datos en tiempo real permite optimizar la producción, aumentar la flexibilidad y personalizar los productos según las demandas del cliente. Al combinar los datos de la planta con la información operativa de la empresa, las fábricas inteligentes logran una mayor transparencia y una toma de decisiones más eficiente.(IBM, s.f)

Industria 4.0

¿Qué es La industria 4.0?

La Industria 4.0 representa la siguiente etapa en la evolución de la manufactura, donde las máquinas no solo ejecutan tareas, sino que también se comunican y colaboran entre sí para desarrollar nuevos productos y servicios. Conocida como la cuarta revolución industrial, esta transformación está redefiniendo la manera en que vivimos y trabajamos, con un impacto significativo en la economía global. Se basa en la digitalización de los procesos industriales mediante tecnologías como el IoT industrial, la automatización, la robótica, el mantenimiento predictivo, la simulación, la fabricación aditiva y el análisis de datos IoT. Su impulso proviene de la necesidad de mejorar la eficiencia, adaptarse rápidamente a las condiciones del mercado, optimizar la calidad y crear nuevos modelos de negocio. El término fue introducido por Klaus Schwab, fundador del Foro Económico Mundial, y hoy en día, las empresas aprovechan los avances tecnológicos de esta revolución para mejorar sus procesos y aumentar su competitividad. (PTC, s.f)

Según SAP Concur, las 8 ventajas de la Industria 4.0 son las siguientes:

1. Creación de nuevos modelos de negocio

En virtud del trabajo disruptivo e innovador, la revolución 4.0 ha promovido profundos cambios en diferentes segmentos por lo que ha sido responsable por la aparición de nuevas oportunidades de negocio, incluso, en sectores tradicionales como el agronegocio.

2. Integración de operaciones en tiempo real

Las fábricas inteligentes se caracterizan por el rápido procesamiento de la información posicionándose, de este modo, como otro diferencial relevante de la industria 4.0.

Mediante la adopción de herramientas tecnológicas de inteligencia artificial, es posible monitorear los equipos de forma remota y en tiempo real aumentando así, la transparencia de los procesos y optimizando la disponibilidad de los datos.

3. Reducción de costos

Los recursos típicos de la industria 4.0 posibilitan que las propias máquinas operen y programen mantenimientos de forma autónoma mitigando la necesidad de contratar personal para realizar manualmente este tipo de actividad.

Como consecuencia, la industria logra reducir el presupuesto orientado al ámbito operacional, enfocar recursos hacia acciones que fomenten la productividad y, por consiguiente, mejorar los resultados del negocio.

4. Economía de energía

Los costos con energía eléctrica configuran uno de los gastos más expresivos a nivel industrial por lo que reducirlos se ha transformado en una obligación para los negocios que desean mejorar su eficiencia y llevar su rentabilidad al siguiente nivel.

Gracias a la tecnología, la mayor parte de los equipos industriales cuentan con sensores que detectan la actividad manteniéndose en pleno funcionamiento mientras sea necesario y cesándola en momentos de ocio.

Asimismo, el avance tecnológico ha favorecido la aparición de equipos cada vez más económicos y movidos a energía renovable como es el caso de la solar.

5. Sostenibilidad

Al economizar energía eléctrica, reducir el uso de insumos en virtud del aumento de la eficiencia y promover el consumo de energía renovable, la industria 4.0 se ha posicionado como una aliada del desarrollo sostenido.

6. Disminución de errores

La reducción de la intervención humana en los procesos disminuye la incidencia de errores en las operaciones industriales, incluso, es posible programar las máquinas para alcanzar un nivel mínimo de eficiencia.

7. Aumento de la productividad

La inversión en tecnología permite que se automaticen diferentes procesos promoviendo, de esta manera, una ejecución más ágil y precisa.

En efecto, la automatización de tareas repetitivas posibilita destinar a los profesionales a realizar tareas más complejas, estratégicas, relacionadas con el core de negocios, que realmente añadan valor para los resultados y que guarden relación con los objetivos del negocio.

8. Optimización de la eficiencia

Además de agilizar las tareas y fomentar la reducción de errores en la ejecución de los procesos, la industria 4.0 promueve la utilización de recursos de manera más consciente, inteligente y económica, lo que resulta en el incremento de la eficiencia operacional y en la mejora de los indicadores de desempeño del negocio.

¿Qué tecnologías hacen posible la Industria 4.0?

La Industria 4.0 es posible gracias al desarrollo de sistemas ciberfísicos, los cuales permiten que los componentes físicos sean controlados por software. Estos sistemas surgen de la integración segura entre máquinas, sistemas de control y programas informáticos. La red industrial juega un papel fundamental en este proceso, ya que facilita la conectividad, la recolección de datos y su transporte hacia las aplicaciones. El análisis de los datos recopilados, tanto en centros de datos como en la nube, permite identificar problemas operativos, optimizar los procesos y minimizar el tiempo de inactividad no planificado. (Cisco, s.f)

IOT

¿Qué es IoT?

IoT (Internet of things/Internet de las cosas) es una arquitectura emergente basada en la Internet global que facilita el intercambio de bienes y servicios entre redes de la cadena de suministro y que tiene un impacto importante en la seguridad y privacidad de los actores involucrados. (Jordy S. & Santiago S. , pag 7)

Beneficios del IoT industrial:

Según PTC, los beneficios del IoT industrial son los siguientes:

Maximiza los ingresos:

Permite reducir el tiempo de inactividad no planificado y superar obstáculos operativos, lo que garantiza un crecimiento continuo de los ingresos. Además, facilita la expansión a nuevos mercados, mejora el rendimiento y abre la posibilidad de implementar nuevos modelos de negocio, como el enfoque de productos como servicio.

Reducir el tiempo de comercialización:

Agiliza el desarrollo de aplicaciones y permite llegar más rápido al mercado. Las plataformas de

IoT industrial posibilitan la modernización de activos heredados, añadiendo nuevas funcionalidades sin necesidad de reemplazar el equipo existente.

Menores costos operativos:

Aprovecha los datos generados por los dispositivos y sistemas conectados para mejorar la productividad y la eficiencia, manteniendo bajos los costos. También ayuda a eliminar cuellos de botella en los procesos, optimizando los recursos.

Mejorar la calidad:

Permite asegurar y escalar las operaciones, productos y servicios. Además, el IoT industrial mejora la calidad del servicio, aumenta la fiabilidad de los sistemas y eleva la satisfacción del cliente mediante procesos más precisos y eficientes.

Las tecnologías que hacen posible el IoT

Según IBM, sobre las tecnologías que hacen posible el Iot, menciona son las siguientes:

Sensores y actuadores:

Estos dispositivos son esenciales en el Internet de las Cosas (IoT), ya que permiten la interacción con el entorno físico. Los sensores detectan cambios ambientales como temperatura, luz, movimiento o presión, mientras que los actuadores realizan acciones físicas, como encender motores o abrir válvulas. La automatización es posible cuando ambos trabajan en conjunto, permitiendo que los sistemas resuelvan tareas sin intervención humana.

Tecnologías de conectividad:

Para que los dispositivos IoT puedan enviar datos a la nube, es fundamental que estén conectados a Internet. Entre las tecnologías más utilizadas para esta comunicación se encuentran Wi-Fi,

Bluetooth, redes celulares, Zigbee y LoRaWAN, cada una adaptada a distintas necesidades en cuanto a alcance, consumo energético y velocidad de transmisión.

Computación en la nube:

La nube es el espacio donde se almacenan, procesan y analizan los datos generados por los dispositivos IoT. Las plataformas de computación en la nube proporcionan las herramientas necesarias para gestionar estos datos y desarrollar aplicaciones que permiten interpretar y actuar sobre la información recopilada.

Análisis de big data:

El gran volumen de información generado por los dispositivos IoT requiere herramientas avanzadas de análisis para extraer información útil e identificar patrones. Este análisis puede incluir el uso de algoritmos de aprendizaje automático, modelos de análisis predictivo y visualización de datos, lo que facilita la toma de decisiones basadas en información precisa.

Tecnologías de seguridad y privacidad:

Con la expansión del IoT, la protección de los dispositivos y de los datos se vuelve crucial. Para garantizar la seguridad y privacidad, se emplean tecnologías como la encriptación, los controles de acceso y los sistemas de detección de intrusiones, que protegen la información frente a amenazas cibernéticas.

Tecnologías de Red IoT:

Las tecnologías de red permiten que los dispositivos [IoT](#) se comuniquen con otros dispositivos, aplicaciones y servicios que se ejecutan en la nube. Internet se basa en protocolos estandarizados para garantizar que la comunicación entre dispositivos heterogéneos sea segura y confiable. Los protocolos estándar especifican las reglas y los

formatos que los dispositivos utilizan para establecer y administrar redes y transmitir datos a través de esas redes. (Anna G. & Jim R., 2020)

Protocolos IoT:

Según Anna Gerber y Jim Romeo (2020) señalan lo siguiente:

MQTT

El Protocolo de Transporte de Telemetría de Cola de Mensajes (MQTT) es un sistema de comunicación basado en un modelo de publicación/suscripción, diseñado para dispositivos con ancho de banda limitado. Es especialmente útil en redes inestables y se emplea comúnmente en sensores y dispositivos móviles debido a su ligereza y eficiencia.

AMQP

El Protocolo Avanzado de Cola de Mensajes (AMQP) es un estándar abierto utilizado en middleware para gestionar la mensajería entre aplicaciones. Es conocido por su fiabilidad y seguridad en el intercambio de datos. Un ejemplo destacado de su implementación es RabbitMQ, una de las plataformas más utilizadas para este protocolo.

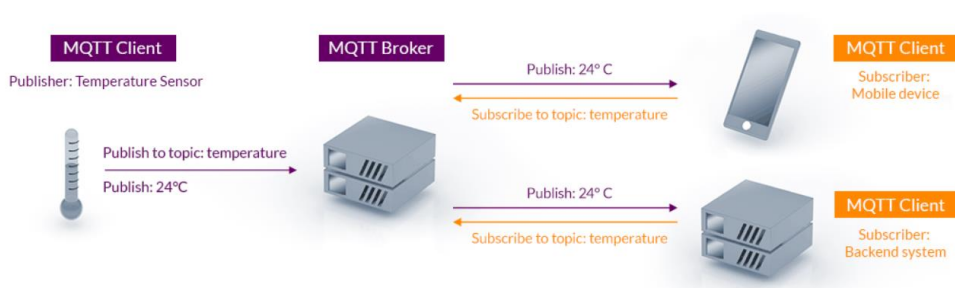
XMPP

El Protocolo Extensible de Mensajería y Presencia (XMPP) fue creado originalmente para la comunicación en tiempo real entre personas, como la mensajería instantánea. Con el tiempo, se ha adaptado para la comunicación máquina a máquina (M2M), permitiendo la gestión de datos XML y la creación de middleware ligero, siendo utilizado principalmente en dispositivos inteligentes.

MQTT

MQTT significa “Message Queue Telemetry Transport” y como su nombre indica se basa en utilizar la mensajería publicación/suscripción entre máquinas en el concepto de “Internet of Things”. Como el protocolo se enfoca, sobre todo, en la comunicación entre sensores, está pensado para ser simple, de fácil implementación, ligero y lo más importante, utilizar muy poco ancho de banda. De esta manera el coste computacional entre la comunicación entre máquinas es bajo y por lo tanto se necesitan pocos recursos (CPU, RAM, etc). Como punto negativo, al utilizar poco ancho de banda y pocos recursos tiene una alta latencia, medida en segundo. Aunque esto no es un problema cuando se trata de sensores ya que para el ojo humano es poco tiempo. Por tanto, los objetivos de la creación del protocolo MQTT es minimizar el ancho de banda, el coste computacional, garantizar la fiabilidad, utilizar comunicación bidireccional entre máquinas y asegurar un grado alto de seguridad. (Mario C.C & Joan G.Ll, pág 15)

Arquitectura de Publicacion/ Suscripción de MQTT



La arquitectura del MQTT se basa en el concepto publicación/suscripción. El “Broker” se encarga de distribuir los mensajes entre los receptores que se hayan suscrito al “topic” (tema en español), de esta manera los receptores (clientes) solo reciben los mensajes que van dirigidos a los “topics” que están suscritos.(Mario C.C & Joan G.Ll, ,pág 17)

Cliente MQTT

En el protocolo MQTT, tanto los suscriptores como los publicadores se consideran clientes MQTT. Un cliente puede desempeñar ambos roles simultáneamente, es decir, puede enviar (publicar) y recibir (suscribirse) a mensajes. Una de las principales ventajas de MQTT es su simplicidad de implementación y su bajo consumo de recursos, lo que permite que funcione en una amplia variedad de dispositivos, desde ordenadores hasta microcontroladores de baja capacidad. (Antonio L, pág 8)

Broker MQTT

El broker MQTT es el servidor responsable de gestionar y distribuir los mensajes entre los dispositivos conectados. Cuando un cliente publica un mensaje, el broker lo recibe, verifica el topic (tema o canal al que está asociado) y lo envía a los clientes suscritos a ese topic. Una de sus características es la posibilidad de almacenar mensajes de forma persistente, lo que permite retener la información hasta que el cliente destinatario se conecte y la reciba. Además, el broker también puede autenticar la identidad de los clientes como una medida de seguridad para proteger la comunicación. (Antonio L, pág 8)

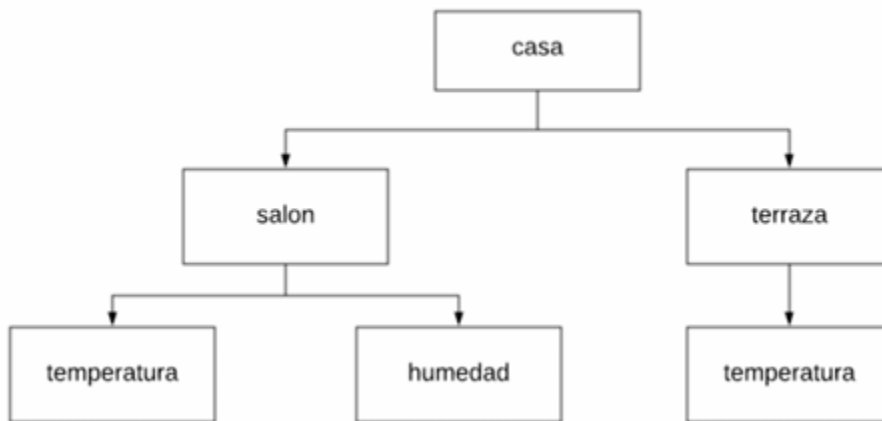
Topic MQTT

El topic es el canal o tema que identifica un mensaje en el protocolo MQTT, indicando a quién está dirigido o de quién se desea recibir información. Sirve como referencia para el broker, permitiéndole enviar los mensajes a los clientes que estén suscritos a ese topic específico. Los topics distinguen entre mayúsculas y minúsculas (*case sensitive*), lo que significa que "Temperatura" y "temperatura" se consideran diferentes. Además, permiten organizarse de forma

jerárquica, facilitando una estructura ordenada para la gestión de múltiples dispositivos o tipos de datos. (Antonio L , pág 9).

Por ejemplo:

- casa/salon/temperatura
- casa/salon/humedad
- casa/terrazza/temperatura



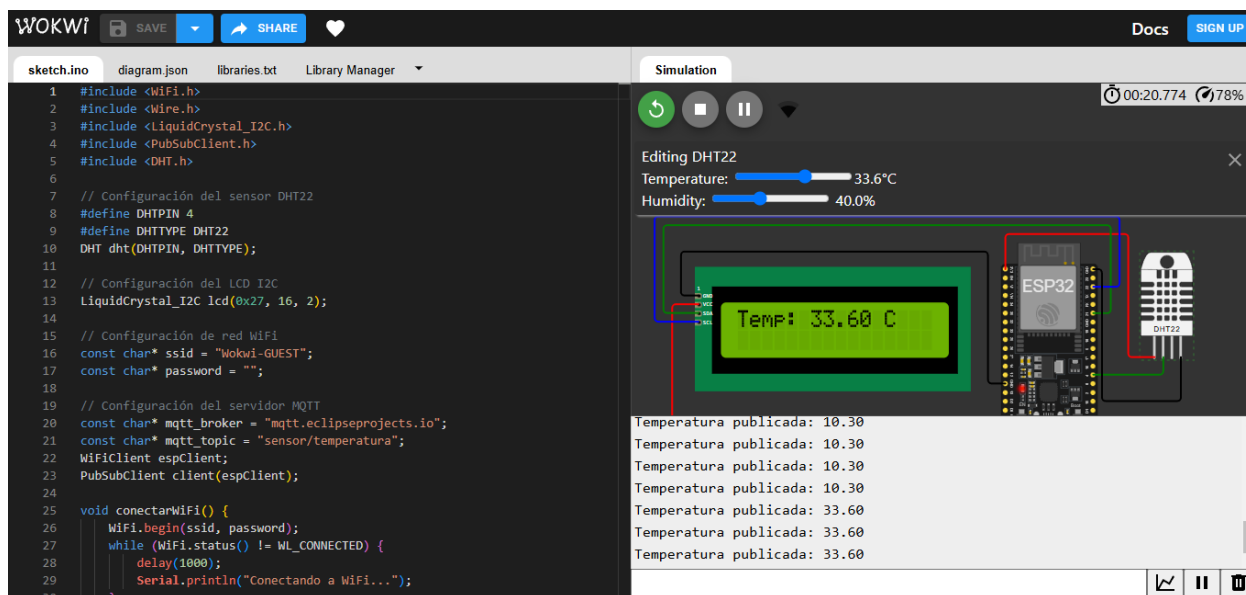
También podemos utilizar los comodines # y + para referirnos a cualquier nivel de la jerarquía.

Ejemplo:

- casa/+ /humedad: sustituye a casa/salon/humedad y casa/cocina/humedad.
- casa/#: engloba todo lo que se encuentre en casa (únicamente se puede utilizar al final).

Circuito Simulado

- **WOKWI**



Wokwi es un simulador en línea que nos permite probar y depurar código para microcontroladores como el **ESP32** sin necesidad de cargarlo en un hardware físico. En el proyecto, Wokwi nos ayuda de varias maneras:

Simulación del ESP32 y el sensor DHT22

- Podemos probar el código de lectura del sensor DHT22 antes de ejecutarlo en un ESP32 real.
- Permite modificar manualmente la temperatura en la interfaz para observar cómo responde el código.

Prueba del protocolo MQTT

- Simula la conexión con un broker MQTT, permitiéndonos verificar si los datos de temperatura se están publicando correctamente en el tópico definido.

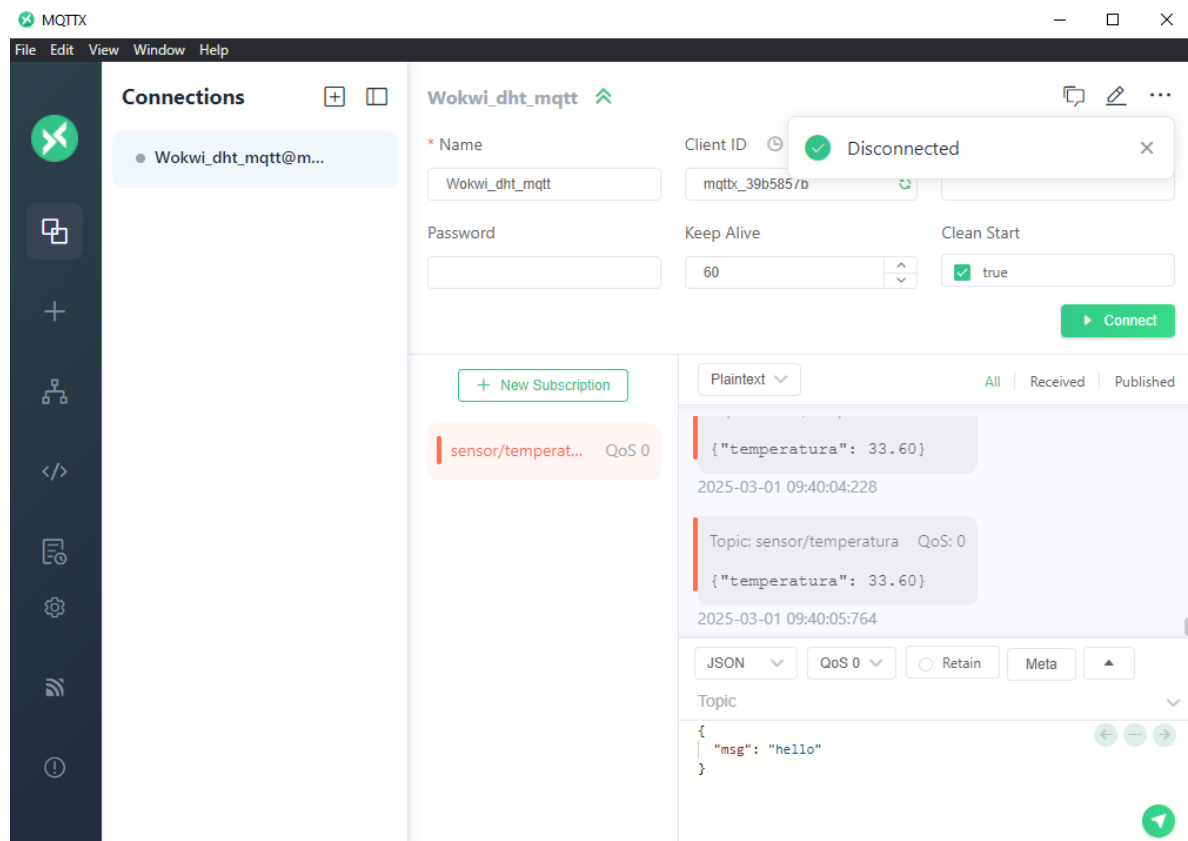
Visualización en LCD

- Podemos comprobar que la temperatura leída del sensor se muestra correctamente en la pantalla LCD sin necesidad de conexiones físicas.

Depuración sin hardware real

- Nos permite corregir errores en el código sin necesidad de conectar un ESP32 real, ahorrando tiempo y evitando problemas con el hardware.

◦ PROGRAMA MQTTX



MQTTX es una herramienta clave para depurar y verificar la comunicación MQTT asegurando que los datos del ESP32 se transmitan correctamente al broker y puedan ser procesados en Python. En el proyecto nos ayuda de varias maneras:

Verificación de mensajes enviados desde el ESP32

- MQTTX nos permite suscribirnos al **tópico "sensor/temperatura"** para recibir los datos que envía el ESP32.
- Podemos ver los valores en tiempo real y comprobar que la temperatura publicada es la esperada.

Detección de errores en la comunicación MQTT

- Si hay problemas de conexión con el broker MQTT (como el mensaje "Disconnected" en la imagen), podemos identificarlos y solucionarlos antes de implementarlo en el hardware real.

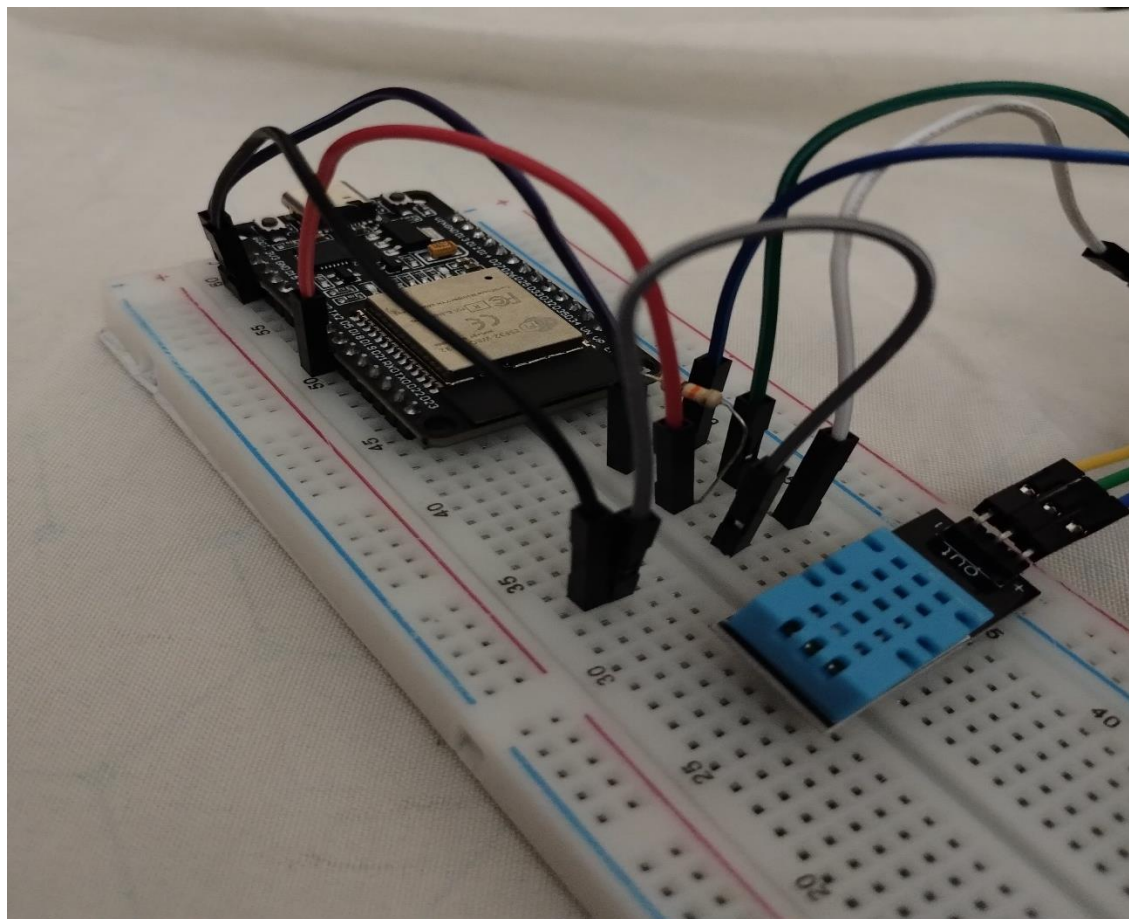
Pruebas de publicación manual

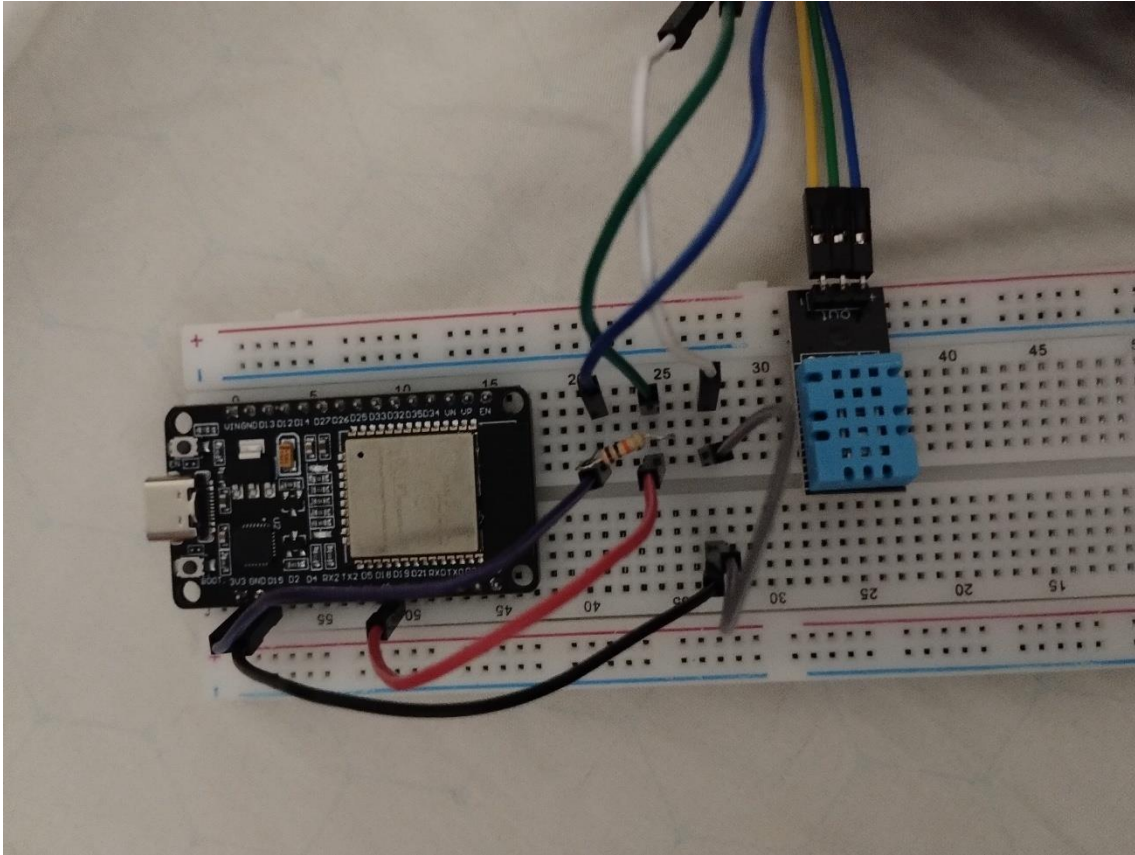
- Podemos enviar datos MQTT manualmente desde MQTTX para simular el comportamiento del ESP32 sin necesidad de ejecutarlo.

Monitoreo de estabilidad y frecuencia de datos

- Observando los mensajes recibidos, podemos verificar si el ESP32 está enviando datos a intervalos correctos y si el formato JSON es adecuado.

Circuito Electrónico





Conexiones observadas:

1. **ESP32 → DHT11:**

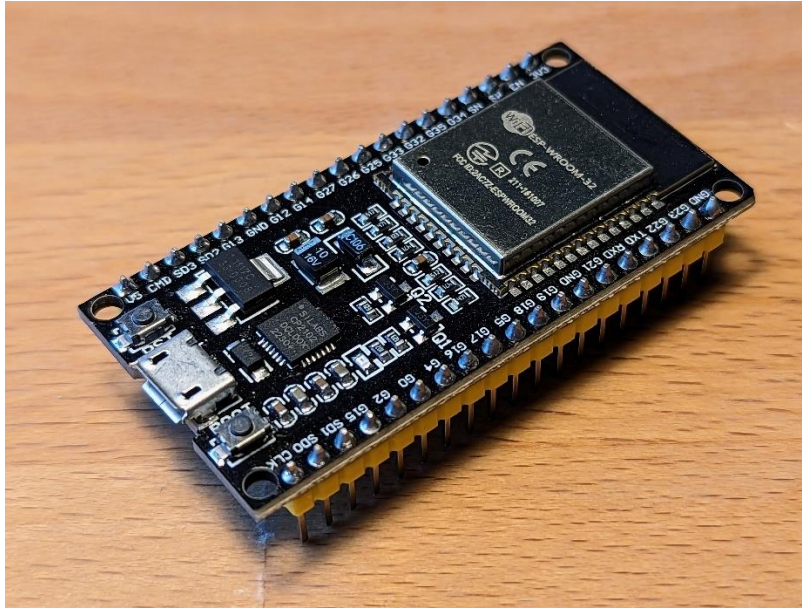
- 3.3V del ESP32 (cable morado) → VCC del DHT11 (primer pin).
- GND del ESP32 (cable negro) → GND del DHT11 (tercer pin).
- GPIO18 del ESP32 (cable rojo) → DATA del DHT11 (segundo pin).

2. **Resistencia Pull-up:**

Hay una resistencia de $10k\Omega$ conectada entre el pin DATA del DHT11 y el 3.3V. Esta resistencia es necesaria para que el ESP32 pueda leer correctamente los datos del sensor.

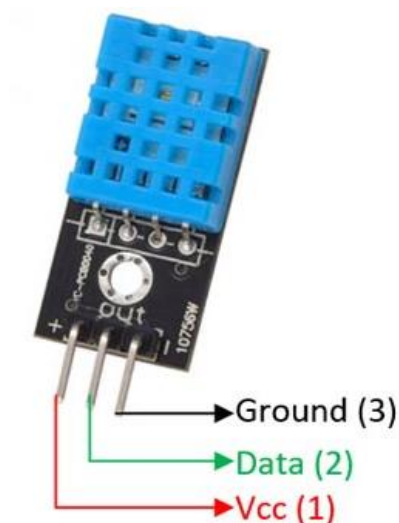
Chip ESP32

El microcontrolador ESP32 tiene funcionalidad Wi-Fi y Bluetooth, comunicación Bluetooth de baja energía (BLE), temporizadores independientes, convertidores de analógico a digital y de digital a analógico (ADC y DAC), sensores táctiles capacitivos y un sensor de efecto Hall. El microcontrolador ESP32 incluye dos núcleos de 240 MHz, cada uno con un microprocesador Tensilica Xtensa LX6 de 32 bits. El microcontrolador ESP32 se incorpora en varios formatos, que van desde una placa de desarrollo hasta un reloj integrado con pantalla táctil y GPS. La variedad de diferentes formatos de ESP32 ilustra la diversidad de proyectos centrados en el microcontrolador ESP32. Los formatos seleccionados son el ESP32 DEVKIT DOIT, el TTGO T-Display V1.1, el TTGO LoRa32 V2.1 1.6, el ESP32-CAM, el TTGO T-Watch V2 con GPS y el módulo M5Stack Core2 (ver Figura 1-1). Todos los formatos incorporan el chip ESP32-D0WDQ6, con revisión 1 (ESP32 DEVKIT DOIT y ESP32-CAM) o revisión 3, excepto el TTGO LoRa32 V2.1 1.6, ya que el chip ESP32-PICO-D4 sustituyó al chip ESP32-D0WDQ6 del TTGO LoRa32 V1.0. La gama seleccionada de formatos de módulos ESP32 no es exhaustiva, pero representa una amplia gama de módulos disponibles con el microcontrolador ESP32. (Neil C., 2023)



Sensor DHT11

El sensor de humedad y temperatura DHT11 es un periférico económico fabricado por D-Robotics UK (www.droboticsonline.com). Es capaz de medir la humedad relativa entre el 20 y el 90 % de humedad relativa dentro del rango de temperatura de funcionamiento de 0 a 50 °C con una precisión del ± 5 % de humedad relativa. La temperatura también se mide en el rango de 0 a 50 °C con una precisión de ± 2 °C. Ambos valores se devuelven con una resolución de 8 bits.(Warren G.,2018)



Pines:

- **VCC (Pin 1):** Alimentación (3.3V o 5V).
- **DATA (Pin 2):** Salida de datos.
- **GND (Pin 3):** Tierra o negativo.

Senamhi

Según el estado peruano, El SENAMHI es un organismo público ejecutor adscrito al Ministerio del Ambiente del Perú. Fue creado el 11 de junio de 1969 y tiene como propósito generar y proveer información y conocimiento meteorológico, hidrológico y climático de manera confiable, oportuna y accesible en beneficio de la sociedad peruana.

Normales climatológicas estándares (CLINO, por sus siglas en inglés):

Son medias/promedios de datos climatológicos calculadas(os) para periodos consecutivos de 30 años, considerando desde el 1 de enero de 1991 hasta el 31 de diciembre de 2020. Las CLINO 1991-2020 calculadas para el Perú cuentan con un récord de 24 a 30 años. SENAMHI (s.f)

Para darle un intervalo a de temperaturas a nuestro sensor DHT11, vamos a obtener un promedio de datos de temperatura acá en el Perú, para eso nos basamos de los datos obtenidos por el SENAMHI, dadas las normales climatológicas obtenidas, son las siguientes:

Enero: 23.446

Febrero: 23.415

Marzo: 23.43

Abril: 23.521

Mayo: 22.983

Junio: 22.207

Julio: 21.973

Agosto: 22.543

Septiembre: 23.123

Octubre: 23.459

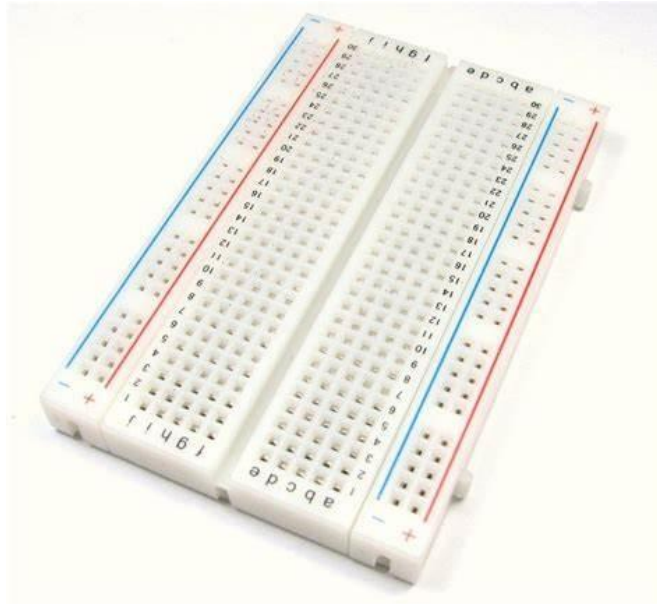
Noviembre: 23.826

Diciembre: 23.581

Teniendo como resultado, un máximo de 23.826 y un mínimo de 21.973. Entonces podríamos considerar respaldándonos de esta información, que, si en caso fuera menor que el mínimo o mayor al máximo, se estaría dando el caso de una anomalía que podría significar una subida o bajada de temperatura fuera de lo habitual.

Protoboard

Distintos métodos de prototipado de circuitos electrónicos se han desarrollado a lo largo del último siglo, donde la experiencia ha seleccionado el proceso más sencillo, económico, práctico y reutilizable, la protoboard, en ella es posible realizar el montaje de circuitos en un corto tiempo y ponerlos a prueba, además, su contribución al sector académico es base de la experticia de un profesional a fin. (Jiménez, W. C. & González, E. S., 2021)



Resistencia

Cuando se le pregunta “¿cual es el funcionamiento de una resistencia, y que es la resistencia Pull-Up?” El texto generado por ChatGPT indico que:

Una resistencia es un componente electrónico que limita o controla el flujo de corriente en un circuito. Su valor se mide en ohmios (Ω).

¿Qué es una resistencia Pull-Up y por qué es importante?

Una resistencia Pull-Up es una resistencia conectada entre un pin de entrada y el voltaje positivo (VCC).

Función principal:

Mantiene el estado lógico "ALTO" (1) cuando no hay señal activa. Esto evita lecturas erráticas o "flotantes" en los microcontroladores.

¿Cómo funciona en tu circuito?

En tu caso, el DHT11 envía datos al ESP32. Cuando el sensor no está enviando información, el pin podría quedar en un estado "flotante" (indefinido). La resistencia Pull-Up:

1. **Cuando no hay señal:** Mantiene el pin en "ALTO" (3.3V).
2. **Cuando hay señal del DHT11:** El sensor "tira" el pin a "BAJO" (0V) para enviar datos.

¿Por qué es crucial la Pull-Up en tu sensor?

Sin esta resistencia, el ESP32 podría recibir datos erróneos o "NaN" (Not a Number) porque el pin no tiene un valor fijo cuando el sensor no transmite.

Jumper

En Electrónica y en particular en informática, un ***jumper*** es un elemento que permite cerrar el circuito eléctrico del que forma parte dos conexiones. Esto puede hacerse mediante soldadura (se derrite suficiente estaño para cerrar el circuito), soldando un cable o alambre entre ambos puntos o, lo más frecuente, conectando dos pines en hilera o paralelo mediante una pieza de plástico que protege el material conductor que cierra el circuito. Los más habituales tienen tamaños de 2,54 mm, 2 mm y 1,27 mm. (Jumper, s.f)



Presupuesto

El Precio de cada objeto que se usó en el proyecto fue:

- PROTOBOARD: S/10.00
- CHIP ESP32: S/32.00
- SENSOR DHT11: S/10.00
- RESISTENCIA 10K Ω : S/0.10
- JUMPER (10 cm): S/4.50

Monto total para realizar el proyecto: S/56.60

Proyectos similares

- **Sistema de Detección de Anomalías para Infraestructuras IOT**

El proyecto propone un sistema de detección de anomalías, centrado en el control de infraestructuras de entornos IoT, lo que se busca es controlar el correcto funcionamiento de estas, permitiendo la toma de acciones casi inmediata tras un aviso de funcionamiento anómalo, o simplemente para su análisis y monitorización diario. Utiliza técnicas avanzadas de Machine Learning para conseguir la detección anomalías.

Características:

- ✓ Seguridad en IoT: Estudia cómo este sistema protege los datos y las infraestructuras.
- ✓ Redes Neuronales aplicadas a la ciberseguridad: Usa técnicas similares para identificar patrones de datos inusuales.



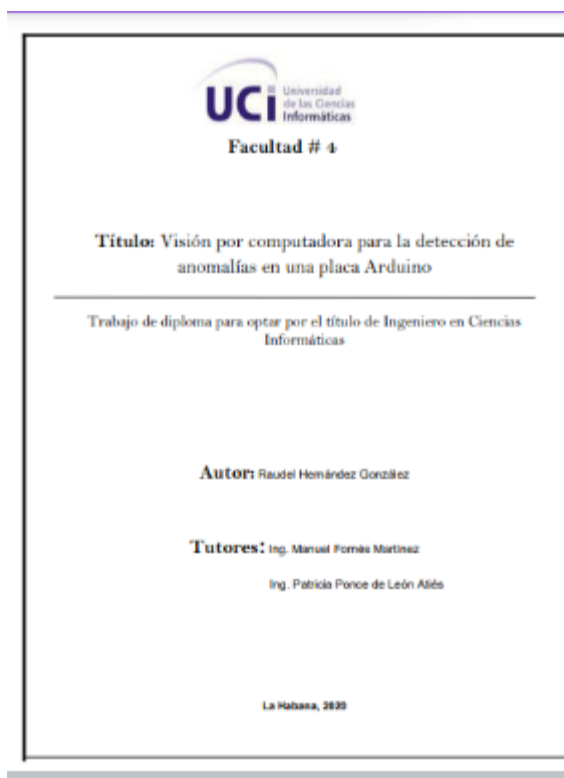
[https://rua.ua.es/dspace/bitstream/10045/135258/1/Sistema de deteccion de anomalias en infraestructuras IoT Arnau Munoz Lucia.pdf](https://rua.ua.es/dspace/bitstream/10045/135258/1/Sistema%20de%20deteccion%20de%20anomalias%20en%20infraestructuras%20IoT%20Arnau%20Munoz%20Lucia.pdf)

- **Visión por Computadora para la detección de Anomalías en una Placa Arduino**

Este proyecto se basa en desarrollar una aplicación que permita la detección de anomalías en una placa Arduino con el uso de visión por computadoras, que permitirá identificar a través de fotos que captura la cámara del teléfono móvil a la placa Arduino posibles anomalías que pueda presentar esta placa.

Características:

- ✓ Uso de redes neuronales: En este proyecto implementan CNN para la detección de anomalías.
- ✓ Visión artificial: Emplean visión por computadora para analizar imágenes.



https://repositorio.uci.cu/bitstream/123456789/10535/1/%E2%80%ABTD_09713_20.pdf

- **Sistema de Alarma para Vivienda con Tecnologías IOT**

Este proyecto tiene como objetivo desarrollar un sistema de alarmas para viviendas, y que les permita detectar situaciones de riesgo o acceso no autorizado en una vivienda, para ello, se empleó un Arduino esp32 junto con varios sensores que sirven para la detección de movimiento, humo, fuego y gas, estos permiten identificar las anomalías en el espacio de la casa en el cual fue instalado.

Características:

- Implementan una prueba piloto del sistema en la vivienda o en las viviendas de los autores.

- Establecen un mecanismo de comunicación entre los dispositivos de detección de intrusos o riesgos y los propietarios de la vivienda



https://ridum.umanizales.edu.co/bitstream/handle/20.500.12746/7090/Gallo_Gonzalez_Andres_Felipe_2024.pdf?sequence=1&isAllowed=y

Tecnologías utilizadas

Librerías usadas:

Arduino:

sensorDHT.ino

```
1  #include <WiFi.h>
2  #include <PubSubClient.h>
3  #include <DHT.h>
```

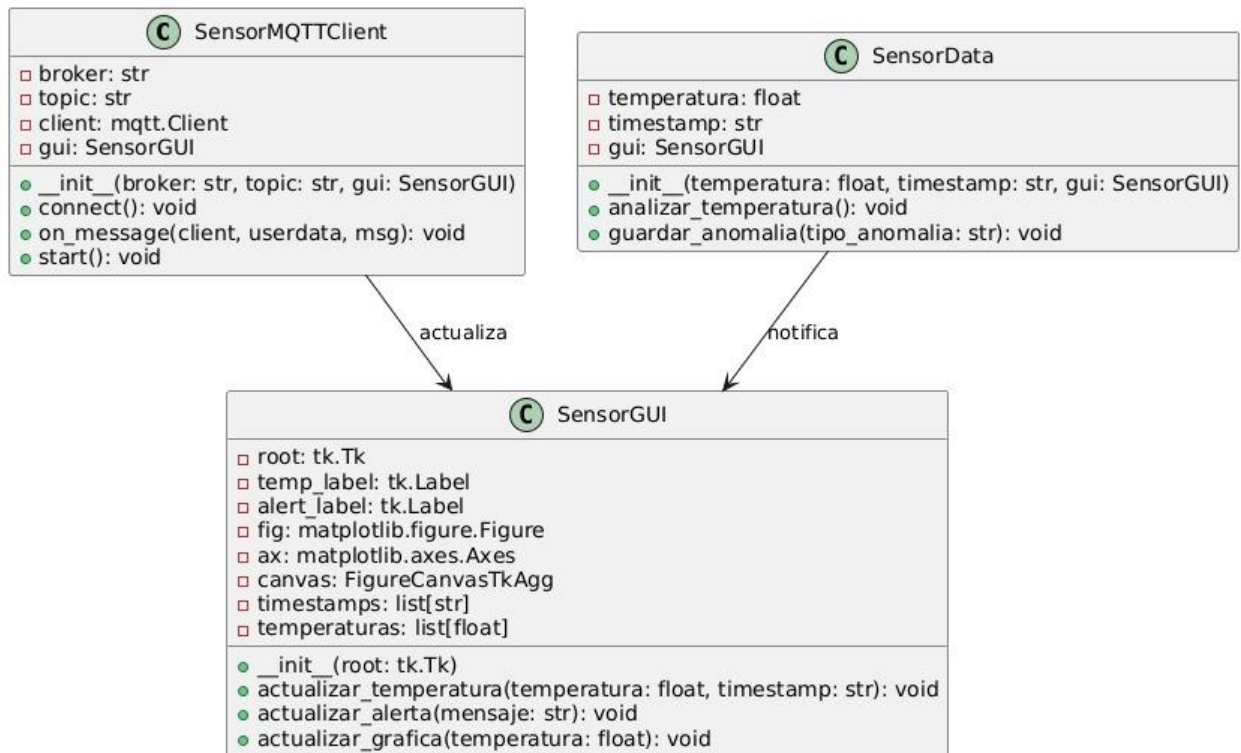
- **WiFi.h:** Librería que permite conectar el **ESP32** a una red WiFi.
- **PubSubClient.h:** Librería que implementa el **protocolo MQTT**, usado para publicar y suscribirse a mensajes.
- **DHT.h:** Librería que facilita la interacción con sensores de temperatura/humedad como **DHT11** o **DHT22**.

Python:

1. **Paho-MQTT (importar paho.mqtt.client como mqtt):** Permite conectar tu programa a un broker MQTT, suscribirse a tópicos y publicar mensajes.
2. **JSON (importar json):** Se usa para trabajar con datos en formato JSON. Permite convertir datos de Python a JSON y viceversa.
3. **Datetime (de datetime importar datetime):** Permite manejar fechas y horas en Python, útil para registrar el momento exacto en que se recibe cada dato.
4. **Tkinter (importar tkinter como tk):** Es la librería estándar de **interfaces gráficas (GUI)** en Python. Se usa para crear ventanas, botones y otros elementos visuales.

5. **Matplotlib (importar matplotlib.pyplot como plt):** Se usa para crear gráficos y visualizaciones en Python. Permite trazar datos en tiempo real o representar tendencias.
6. **Backends de Matplotlib (de matplotlib.backends.backend_tkagg, importe FigureCanvasTkAgg):** Es un módulo de Matplotlib que permite integrar gráficos dentro de interfaces gráficas hechas con Tkinter.
7. **Ticker de Matplotlib (de matplotlib.ticker, importe MaxNLocator):** Se usa para personalizar los ejes de los gráficos, controlando cuántas marcas o etiquetas aparecen.
8. **Numpy (import numpy as np):** Es una librería matemática para manejar arreglos y realizar cálculos como media y desviación estándar, útil en tu diagrama de control.

UML



Estructura de cada clase en el UML:

1. **Parte superior:** Nombre de la clase.
2. **Parte media:** Atributos (variables que almacena la clase).
3. **Parte inferior:** Métodos (funciones que realiza la clase).

1. Relación: **SensorMQTTClient** → **SensorGUI**

- Tipo de Relación:

Asociación (una clase usa a otra sin poseerla de forma permanente).

- Naturaleza de la Asociación:

Direccional: **SensorMQTTClient** depende de **SensorGUI**.

Débil (no permanente): `SensorMQTTClient` solo mantiene una referencia a `SensorGUI`, pero no la posee.

- Flujo de Comunicación:

1. `SensorMQTTClient` recibe un mensaje MQTT.
2. Decodifica el mensaje y crea un objeto `SensorData`.
3. Utiliza `self.gui` para actualizar la interfaz gráfica con los datos recibidos

- Propósito de la Asociación:

- ✓ Sincronizar la recepción de datos en tiempo real con la interfaz gráfica.
- ✓ Mantener separado el procesamiento de datos (MQTT) de su visualización (GUI).

2. Relación: `SensorMQTTClient` → `SensorData`

- Tipo de Relación:

Composición (una clase posee a otra y controla su ciclo de vida).

- Naturaleza de la Asociación:

- ✓ Unidireccional: `SensorMQTTClient` crea y gestiona `SensorData`.
- ✓ Fuerte (permanente mientras dura el procesamiento): Cada mensaje MQTT genera un nuevo objeto `SensorData`, que es descartado al finalizar su uso.

- Explicación:

Cada vez que SensorMQTTClient recibe un mensaje MQTT, instancia un objeto SensorData para procesarlo

- Flujo de Comunicación:

- ✓ SensorMQTTClient recibe un mensaje MQTT.
- ✓ Crea un objeto SensorData para analizar la temperatura.
- ✓ SensorData evalúa la temperatura y notifica a SensorGUI si detecta una anomalía.

- Propósito de la Asociación:

- ✓ Delegar el análisis de temperatura a una clase especializada.
- ✓ Facilitar la organización del código mediante la separación de responsabilidades.

3. Relación: SensorData → SensorGUI

- Tipo de Relación:

Asociación (una clase notifica a otra).

- Naturaleza de la Asociación:

- ✓ Unidireccional: SensorData informa a SensorGUI.
- ✓ Débil: SensorData usa SensorGUI, pero no la controla ni la posee.

- **Explicación:**

Cuando SensorData detecta un cambio en la temperatura, actualiza la interfaz gráfica

- **Flujo de Comunicación:**

1. SensorData analiza la temperatura.
2. Si detecta una anomalía, actualiza la interfaz gráfica (SensorGUI).
3. Además, guarda el evento en un archivo JSON para su posterior análisis.

- **Propósito de la Asociación:**

- ✓ Actualizar la interfaz gráfica con los resultados del análisis.
- ✓ Informar al usuario si se detectan anomalías en la temperatura

4. Relación: SensorGUI ↔ SensorMQTTClient

- **Tipo de Relación:**

Asociación bidireccional (ambas clases dependen entre sí).

- **Naturaleza de la Asociación:**

- ✓ Bidireccional: SensorMQTTClient necesita a SensorGUI y viceversa.
- ✓ Débil: Ninguna clase es propietaria de la otra; solo comparten referencias.

- **Explicación:**

Al inicializar SensorMQTTClient, se le pasa una referencia a la GUI

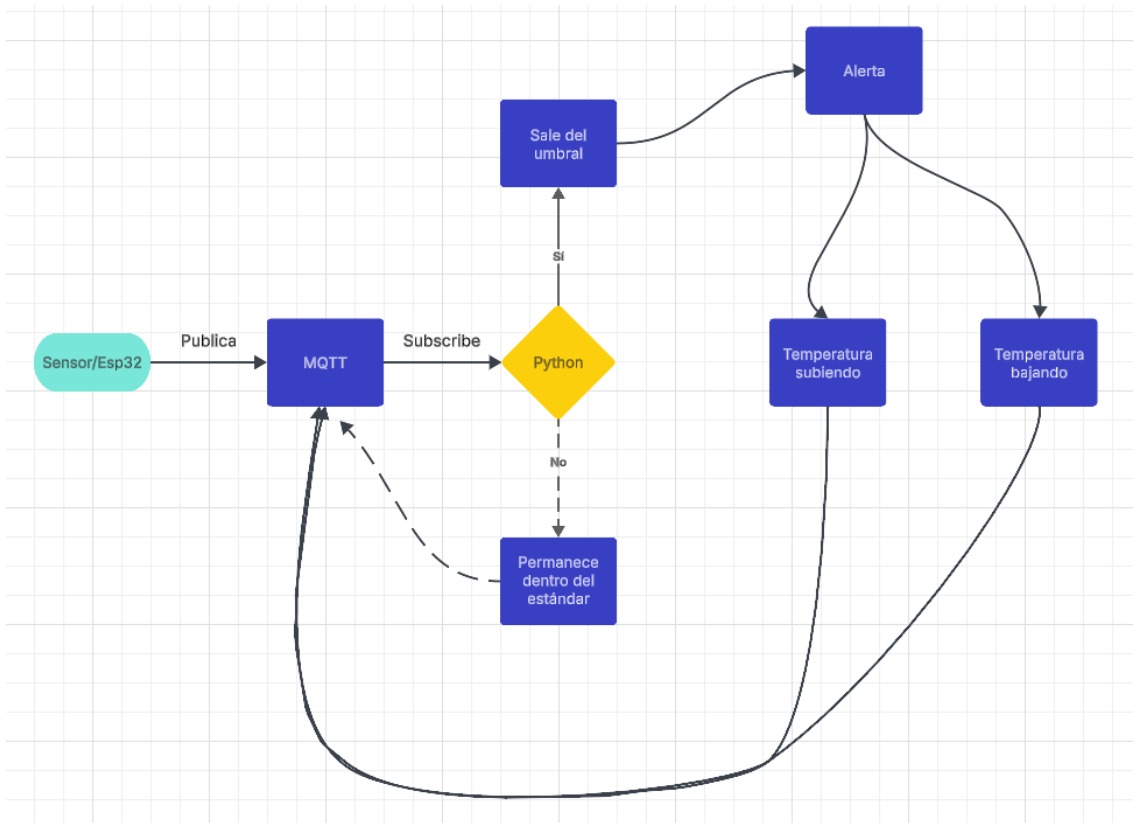
- **Flujo de Comunicación:**

1. Entrada: SensorMQTTClient recibe datos del sensor.
2. Procesamiento: SensorData analiza la temperatura y detecta posibles anomalías.
3. Salida: SensorGUI actualiza la interfaz para mostrar los resultados.

- **Propósito de la Asociación:**

- ✓ Sincronizar la recepción de datos (MQTT) con su visualización (GUI).
- ✓ Permitir una actualización en tiempo real de la información.

Diagrama de bloques



1. Descripción General del Flujo

1. Sensor (DHT11):

- El sensor DHT11 mide la temperatura en tiempo real.
- Estos valores de temperatura se envían al ESP32 para ser procesados.

2. Publicación en MQTT:

- El ESP32 publica los valores de temperatura al broker MQTT en un tópico específico (por ejemplo, temperatura/sensor).
- El broker MQTT es responsable de distribuir los mensajes a cualquier suscriptor interesado.

3. Suscripción en Python:

- Un programa en Python está suscrito a este tópico MQTT.
- Cada vez que se recibe un nuevo valor de temperatura, se procesa para analizar si es una anomalía o no.

2. Análisis de Temperatura en Python

En este paso, el código en Python realiza las siguientes acciones:

1. Compara la temperatura con los umbrales predefinidos:

- LSC (Límite Superior de Control): Temperatura máxima aceptable.
- LIC (Límite Inferior de Control): Temperatura mínima aceptable.

2. Decisión:

- Si la temperatura está fuera del umbral (anomalía):
 - Se genera una alerta.
 - Se analiza si la temperatura es mayor o menor que el rango permitido.
- Si la temperatura permanece dentro del estándar, se continúa el monitoreo normal.

3. Manejo de Anomalías

Cuando se detecta una anomalía:

1. Generar una Alerta:

- Si la temperatura está fuera de los límites, se activa una alerta.

2. Determinar la Dirección del Cambio:

- Si la temperatura es mayor al umbral superior, se indica "Temperatura subiendo".
- Si la temperatura es menor al umbral inferior, se indica "Temperatura bajando".

4. Bucle de Monitoreo Continuo

El proceso funciona en un bucle infinito:

- El ESP32 sigue midiendo la temperatura y enviándola al broker MQTT.
- El programa en Python recibe los valores, los analiza y genera alertas si es necesario.
- El sistema continúa en monitoreo continuo, evaluando cada nuevo dato recibido.

5. Flujo de Decisión en el Diagrama

1. **Sensor publica** → Los datos de temperatura se envían al broker MQTT.
2. **Python se suscribe** → Recibe los datos de temperatura del broker.
3. **Condición (¿Sale del umbral?):**

Sí:

- Genera una alerta.
- Evalúa si la temperatura está subiendo (supera el LSC) o bajando (cae por debajo del LIC).

No:

- La temperatura está dentro del rango normal.

2. **Repetición del ciclo** → El sistema sigue monitoreando de forma continua.

Código Arduino

En esta sección pasaremos a explicar paso a paso el código que se usó en el Arduino IDE:

Configuración WIFI:

```

5  const char* ssid = "CCFFAA LIMA-2.4G";
6  const char* password = "191114$RAMI";

```

- **ssid**: Nombre de la red WiFi a la que se conectará el ESP32.
- **password**: Contraseña de la red.

Importante: Asegúrate de cambiar estas credenciales si cambias de red.

Configuración de Broker MQTT:

```

8  const char* mqttServer = "broker.emqx.io";
9  const int mqttPort = 1883;
10 const char* topic_temp = "temperatura/sensor";

```

- **mqttServer**: Dirección del **broker MQTT** (en este caso, **broker.emqx.io**, un broker público gratuito).
- **mqttPort**: Puerto **1883** es el estándar para conexiones MQTT sin encriptación.
- **topic_temp**: Es el **tópico** donde el **ESP32** enviará las mediciones de temperatura. Los suscriptores a este tópico recibirán los valores.

Configuración de Sensor DHT11:

```

12 #define DHTPIN 18
13 #define DHTTYPE DHT11
14 DHT dht(DHTPIN, DHTTYPE);

```

- **DHTPIN**: El **GPIO 18** es el pin del **ESP32** donde está conectado el **DHT11**.
- **DHTTYPE**: Especifica el **modelo del sensor** (DHT11 o DHT22).
- **DHT dht**: Crea un **objeto** dht que se usará para leer los datos del sensor.

Creación de objetos para la creación:

```

16 WiFiClient espClient;
17 PubSubClient mqttClient(espClient);

```

- **WiFiClient**: Objeto para gestionar la conexión WiFi.
- **PubSubClient**: Objeto para gestionar la comunicación con el **broker MQTT**.

Función para conectar a la red Wifi:

```

19 void setupWifi() {
20     delay(1000);
21     Serial.print("Conectando a WiFi");
22     WiFi.begin(ssid, password);
23
24     while (WiFi.status() != WL_CONNECTED) {
25         delay(500);
26         Serial.print(".");
27     }
28     Serial.println("\nWiFi conectado.");
29 }

```

- **delay(1000):** Espera 1 segundo antes de iniciar la conexión.
- **WiFi.begin(ssid, password):** Intenta conectar con las credenciales definidas.
- **while (WiFi.status() != WL_CONNECTED):** Bucle que espera hasta que se establezca la conexión.
- **Serial.print("."): Imprime un punto cada 500 ms mientras intenta conectar.**
- **Serial.println("\nWiFi conectado.):** Informa cuando la conexión es exitosa.

Función para conectar al bróker mqtt:

```

31 void connectToMqtt() {
32     while (!mqttClient.connected()) {
33         Serial.println("Conectando al broker MQTT...");
34
35         if (mqttClient.connect("ESP32_Client")) {
36             Serial.println("Conectado a MQTT.");
37         } else {
38             Serial.print("Fallo en la conexión MQTT. Estado: ");
39             Serial.println(mqttClient.state());
40             delay(2000);
41         }
42     }
43 }

```

- **mqttClient.connected()**: Verifica si el cliente ya está conectado.
- **mqttClient.connect("ESP32_Client")**: Intenta conectar al **broker MQTT** con el nombre "ESP32_Client".
- **mqttClient.state()**: Muestra un código de error si la conexión falla.
- **delay(2000)**: Espera 2 segundos antes de volver a intentar conectar.

Función de inicialización:

```

45 void setup() {
46     Serial.begin(115200);
47     delay(5000);
48
49     dht.begin();
50     setupWifi();
51
52     mqttClient.setServer(mqttServer, mqttPort);
53     connectToMqtt();
54 }

```

- **Serial.begin(115200):** Inicia la comunicación serial a 115200 baudios (velocidad de transmisión).
- **delay(5000):** Espera 5 segundos para asegurar lecturas estables del DHT.
- **dht.begin():** Inicializa el sensor DHT.
- **setupWifi():** Conecta a la red WiFi.
- **mqttClient.setServer():** Establece el **broker MQTT** y el puerto.
- **connectToMqtt():** Conecta al broker MQTT.

Bucle principal (loop()):

```

56 void loop() {
57     if (!mqttClient.connected()) {
58         connectToMqtt();
59     }
60
61     mqttClient.loop();
62
63     float temperatura = NAN;
64     for (int i = 0; i < 3 && isnan(temperatura); i++) {
65         temperatura = dht.readTemperature();
66         delay(100); // Pequeña espera entre intentos
67     }
68
69     if (isnan(temperatura)) {
70         Serial.println("Error leyendo el sensor DHT22 después de 3 intentos");
71     } else {
72         char tempStr[50];
73         snprintf(tempStr, 50, "{\"temperatura\": %.2f}", temperatura);
74         mqttClient.publish(topic_temp, tempStr);
75
76         Serial.print("Temperatura publicada: ");
77         Serial.println(tempStr);
78     }
79
80     delay(2000);

```

- **Reconecta** al broker MQTT si se pierde la conexión.
- **Lee la temperatura** (hasta 3 intentos si falla la lectura).
- **Publica la temperatura** en formato JSON ({\"temperatura\": valor}).

- **Muestra en el monitor serial** el valor publicado o el error si no hay lectura.

Código Python

El proyecto de software “SHIELDNET: Sistema Inteligente de Detección y Protección en redes IOT” utiliza diversas librerías y herramientas que hacen posible su correcto funcionamiento e interpretación del código en Python y así lograr que dicho programa pueda suscribirse a los tópicos del broker MQTT, reciba los datos de temperatura de la placa ESP32 y los pueda analizar en tiempo real. A continuación, se mencionarán las librerías empleadas y el objetivo de éstas en el código del proyecto:

```
import paho.mqtt.client as mqtt
import json
from datetime import datetime
import tkinter as tk
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from matplotlib.ticker import MaxNLocator
import numpy as np # Importamos numpy para calcular media y desviación estándar
```

9. Paho-MQTT (importar paho.mqtt.client como mqtt):

Esta librería es fundamental en este proyecto, ya que permite la conexión con un broker MQTT, la suscripción a un tópico de temperatura y el procesamiento de los datos enviados desde el ESP32. Además, facilita la recepción de mensajes en tiempo real, asegurando una actualización continua de la información.

10. JSON (importar json):

En este proyecto, se utiliza para decodificar los mensajes MQTT y para guardar anomalías detectadas en un archivo JSON, garantizando un registro estructurado de eventos importantes.

11. Datetime (de datetime importar datetime):

El módulo datetime proporciona herramientas para manejar fechas y horas. Su implementación en este código permite registrar la fecha y hora exacta de cada temperatura recibida.

12. Tkinter (importar tkinter como tk):

Con esta biblioteca se pueden desarrollar interfaces gráficas de usuario (GUI). En este proyecto, se emplea para diseñar una ventana de monitoreo, donde se muestra en tiempo real la temperatura recibida y las alertas en caso de anomalías.

13. Matplotlib (importar matplotlib.pyplot como plt):

Matplotlib es una biblioteca especializada en la generación de gráficos. Su uso en este proyecto permite representar visualmente la temperatura recibida, trazando un gráfico de control con los valores históricos.

14. Backends de Matplotlib (de matplotlib.backends.backend_tkagg, importe FigureCanvasTkAgg):

Este módulo permite la integración de gráficos de Matplotlib dentro de la interfaz gráfica de Tkinter. FigureCanvasTkAgg, el gráfico de control de temperatura.

15. Ticker de Matplotlib (de matplotlib.ticker, importe MaxNLocator):

En este proyecto, MaxNLocator(nbins=5) se emplea para ajustar la cantidad de etiquetas en el eje X del gráfico, debido a que se encarga de personalizar la visualización de los ejes en los gráficos, mejorando la legibilidad y evitando el amontonamiento de los datos.

16. Numpy (import numpy as np):

Esta librería es importante para realizar la gráfica de control, ya que nos ayuda a calcular la media y la desviación estándar de las temperaturas registradas.

En este proyecto también se han empleado algunas clases que mejoran el entendimiento del código manteniendo una organización, para así poder distribuir cada tarea o función en bloques organizados que pueden ser modificados fácilmente sin tener que volver a analizar todo el código. Estas clases contienen un enfoque orientado a objetos, del cual se basa este curso de programación, permitiéndonos mayor claridad para su manejo y fácil edición del código. A continuación, se presentarán todas las clases usadas y detallaremos el funcionamiento del contenido de estas:

1. Clase SensorMQTTClient:

Esta clase se encarga de manejar la comunicación con el broker MQTT, recibir los datos enviados desde el ESP32 y procesarlos.

a) Constructor `__init__` :

```
class SensorMQTTClient:
    def __init__(self, broker, topic, gui):
        self.broker = broker
        self.topic = topic
        self.client = mqtt.Client()
        self.client.on_message = self.on_message
        self.gui = gui
```

- **Bróker:** Indica la dirección del servidor MQTT al que se conectará (en este caso, "mqtt.eclipseprojects.io").
- **Topic:**ópico MQTT al que se suscribe para recibir los datos (en este caso, "sensor/temperatura").
- **self.client = mqtt.Client():** Crea un cliente MQTT utilizando la librería paho.mqtt.client.

- **self.client.on_message = self.on_message:** Asigna la función on_message para manejar los mensajes que llegan desde el ESP32.
- **self.gui = gui:** Referencia a la interfaz gráfica SensorGUI, para actualizar la GUI en función de los datos recibidos.

b) Método connect:

```
def connect(self):
    self.client.connect(self.broker, 1883, 60)
    self.client.subscribe(self.topic)
    print(f"Conectado al broker {self.broker}, suscrito al tópico {self.topic}")
```

- **self.client.connect(self.broker, 1883, 60):** Va a establecer la conexión con el **broker MQTT** en el puerto **1883** (el estándar para MQTT), sabiendo además que el 60 representa el tiempo máximo de espera para la reconexión en caso de desconexión.
- **self.client.subscribe(self.topic):** Se suscribe al **tópico MQTT** donde se reciben los datos de temperatura.

c) Método on_message (Procesamiento de datos recibidos): Este método se activa cada vez que llega un mensaje al tópico MQTT.

```
def on_message(self, client, userdata, msg):
    try:
        data = json.loads(msg.payload.decode().strip())
        timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
        sensor_data = SensorData(float(data.get("temperatura", 0)), timestamp, self.gui)
        sensor_data.analizar_temperatura()
    except Exception as e:
        print("Error procesando mensaje:", e)
```

- **client:** Cliente MQTT que recibió el mensaje.
- **msg:** Contiene el mensaje MQTT recibido.

- Decodifica el mensaje MQTT de **JSON** a **diccionario**.
- Obtiene la temperatura y la almacena con la hora actual.
- Llama a **analizar_temperatura()** para verificar si hay anomalías.

d) Método start:

Mantiene la conexión con el broker para recibir datos continuamente.

```
def start(self):
    self.client.loop_forever()
```

2. Clase **SensorData** (Procesamiento de la temperatura):

Analiza la temperatura, detecta anomalías y las guarda en un archivo.

a) Constructor `__init__`:

```
class SensorData:
    def __init__(self, temperatura, timestamp, gui):
        self.temperatura = temperatura
        self.timestamp = timestamp
        self.gui = gui

    def analizar_temperatura(self):
        print(f"[{self.timestamp}] Temperatura recibida: {self.temperatura}°C")
        self.gui.actualizar_temperatura(self.temperatura, self.timestamp)
        self.gui.actualizar_grafica(self.temperatura)
```

- **temperatura:** Valor recibido del sensor.
- **timestamp:** Fecha y hora de la medición.
- **gui:** Referencia a la interfaz gráfica.

```

if len(self.gui.temperaturas) >= 10:
    mu = np.mean(self.gui.temperaturas)
    sigma = np.std(self.gui.temperaturas)
    LIMITE_SUPERIOR = mu + 3 * sigma
    LIMITE_INFERIOR = mu - 3 * sigma

    if self.temperatura > LIMITE_SUPERIOR or self.temperatura < LIMITE_INFERIOR:
        tipo_anomalia = "Temperatura alta" if self.temperatura > LIMITE_SUPERIOR else "Temperatura baja"
        print(f"▲ Anomalia detectada: {tipo_anomalia} ({self.temperatura}°C)")
        self.guardar_anomalia(tipo_anomalia)

```

Esta parte del código implementa un control estadístico de calidad para la temperatura:

- Primero, espera tener al menos 10 datos para hacer un análisis válido.
- Luego, calcula los límites de control dinámicos basados en la media y la desviación estándar.
- Si una temperatura medida se sale de estos límites, la detecta como anómala y la guarda en un archivo JSON para su posterior análisis.

Este enfoque evita el uso de valores límite fijos y permite que el sistema se adapte automáticamente a los datos recibidos.

b) Método guardar_anomalia:

```

def guardar_anomalia(self, tipo_anomalia):
    """Guarda las anomalías detectadas en un archivo JSON."""
    anomalia = {
        'timestamp': self.timestamp,
        'temperatura': self.temperatura,
        'tipo_anomalia': tipo_anomalia
    }
    try:
        archivo = 'anomalias_detectadas.json'
        datos = []

        if os.path.exists(archivo):
            with open(archivo, 'r') as file:
                datos = json.load(file)
                if not isinstance(datos, list):
                    datos = []

            datos.append(anomalia)

        with open(archivo, 'w') as file:
            json.dump(datos, file, indent=4) # Guardamos la anomalía con formato JSON

        print(f"Anomalia guardada en el archivo: {anomalia}")
    except Exception as e:
        print(f"Error al guardar la anomalía: {e}")

```

Este fragmento de código define el método `guardar_anomalia(self, tipo_anomalia)`, que almacena anomalías detectadas en un archivo JSON llamado **"anomalías_detectadas.json"**

Crea un diccionario (anomalía) con la información de la anomalía detectada:

- `timestamp`: Marca de tiempo del evento.
- `temperatura`: Valor de temperatura en el momento de la anomalía.
- `tipo_anomalia`: Tipo de anomalía detectada.

Verifica si el archivo JSON existe:

- Si existe, lo abre y carga los datos previos.
- Si los datos no son una lista, inicializa una lista vacía.
- Si el archivo no existe, crea una lista vacía.

Añade la nueva anomalía a la lista cargada.

Escribe la lista actualizada en el archivo JSON, con formato legible (`indent=4`).

Manejo de errores: Si ocurre un error en el proceso, se captura la excepción y se muestra un mensaje de error.

c) Constructor `__init__`:

```
class SensorGUI:
    def __init__(self, root):
        self.root = root
        self.root.title("Monitoreo de Temperatura")
        self.root.geometry("600x400")

        self.temp_label = tk.Label(root, text="Temperatura: --°C", font=("Arial", 14))
        self.temp_label.pack(pady=20)

        self.alert_label = tk.Label(root, text="Estado: Esperando datos...", font=("Arial", 12))
        self.alert_label.pack(pady=10)

        self.fig, self.ax = plt.subplots(figsize=(5, 3))
        self.ax.set_title("Gráfica de Control - Temperatura")
        self.ax.set_xlabel("Tiempo")
        self.ax.set_ylabel("Temperatura (°C)")
        self.ax.set_ylim(20, 30)

        self.timestamps = []
        self.temperaturas = []

        self.canvas = FigureCanvasTkAgg(self.fig, master=root)
        self.canvas.get_tk_widget().pack(pady=20)
```

- Recibe **root**, que es la ventana principal de Tkinter.
- Define el **título** de la ventana como "Monitoreo de Temperatura".
- Ajusta el **tamaño** de la ventana a **600x400 píxeles**.
- Crea etiquetas (Label) para mostrar la temperatura y el estado del sensor.
- Configura la gráfica de temperatura con un título, ejes y límites predefinidos.
- Crea listas para almacenar los datos y un canvas para integrar la gráfica en Tkinter.

d) Método `actualizar_temperatura`:

```
def actualizar_temperatura(self, temperatura, timestamp):
    self.temp_label.config(text=f"Temperatura: {temperatura}°C")
```

Muestra la temperatura actual en la ventana.

e) Método actualizar_alerta:

```
def actualizar_alerta(self, mensaje):
    self.alert_label.config(text=f"Estado: {mensaje}")
```

Actualiza la alerta según el estado de la temperatura (normal, alta o baja).

f) Método actualizar_grafica:

```
def actualizar_grafica(self, temperatura):
    self.timestamps.append(datetime.now().strftime("%H:%M:%S"))
    self.temperaturas.append(temperatura)

    if len(self.timestamps) > 10:
        self.timestamps.pop(0)
        self.temperaturas.pop(0)

    self.ax.clear()
    self.ax.set_title("Gráfica de Control - Temperatura")
    self.ax.set_xlabel("Tiempo")
    self.ax.set_ylabel("Temperatura (°C)")

    if len(self.temperaturas) >= 10:
        mu = np.mean(self.temperaturas)
        sigma = np.std(self.temperaturas)
        LIMITE_SUPERIOR = mu + 3 * sigma
        LIMITE_INFERIOR = mu - 3 * sigma
    else:
        LIMITE_SUPERIOR, LIMITE_INFERIOR = None, None

    self.ax.plot(self.timestamps, self.temperaturas, marker='o', color='blue', label='Temperatura')

    if LIMITE_SUPERIOR is not None and LIMITE_INFERIOR is not None:
        self.ax.axhline(y=LIMITE_SUPERIOR, color='red', linestyle='--', label=f'UCL ({LIMITE_SUPERIOR:.2f}°C)')
        self.ax.axhline(y=LIMITE_INFERIOR, color='green', linestyle='--', label=f'LCL ({LIMITE_INFERIOR:.2f}°C)')

    self.ax.set_ylim(20, 30)
    self.ax.xaxis.set_major_locator(MaxNLocator(nbins=5))
    self.ax.set_xticks(range(len(self.timestamps)))
    self.ax.set_xticklabels(self.timestamps, rotation=45, ha='right')
```

Este método se encarga de actualizar la gráfica de control de temperatura en la interfaz gráfica, incluyendo los límites de control dinámicos basados en la media y la desviación estándar de los últimos 10 datos.

- Limpia la gráfica anterior y dibuja la nueva curva de temperatura.

- **Dibuja líneas que marcan los límites superior e inferior.**
- Ajusta el formato del eje X y redibuja la gráfica en la interfaz.

Calcula los límites de control dinámicamente (si hay al menos 10 datos):

- Se obtiene la media (μ) y la desviación estándar (σ) de los últimos 10 valores.
- Se calculan los límites, superior ($LSC = \mu + 3 * \sigma$) e inferior ($LIC = \mu - 3 * \sigma$).

3. Código Principal:

```
if __name__ == "__main__":
    root = tk.Tk()
    gui = SensorGUI(root)
    mqtt_client = SensorMQTTClient("mqtt.eclipseprojects.io", "sensor/temperatura", gui)
    mqtt_client.connect()

    import threading
    mqtt_thread = threading.Thread(target=mqtt_client.start)
    mqtt_thread.daemon = True
    mqtt_thread.start()

    root.mainloop()
```

Es el punto de entrada del programa, donde se inicializa todo.

- Crea la ventana principal y la interfaz gráfica.
- Inicializa la conexión MQTT para recibir los datos del sensor.
- Ejecuta MQTT en un hilo separado para no bloquear la interfaz gráfica.
- Mantiene la ventana abierta y procesando eventos.

Conclusiones

- El proyecto *ShieldNet* integra de manera eficiente el monitoreo y análisis de datos en tiempo real utilizando tecnologías IoT. La combinación del **ESP32**, el **sensor DHT11** y el protocolo **MQTT** permite capturar, transmitir y procesar las lecturas de temperatura de forma continua. Esta arquitectura garantiza que cualquier

anomalía en la temperatura sea detectada rápidamente, activando alertas específicas según el comportamiento del sistema. Además, la implementación de un sistema de reconexión automática a la red WiFi y al broker MQTT fortalece la estabilidad del proyecto, asegurando la disponibilidad del monitoreo incluso en condiciones de conexión inestable.

- El diseño modular del sistema permite una integración sencilla con otros sensores y dispositivos IoT, lo que facilita futuras expansiones del proyecto *ShieldNet*. El uso de Python para analizar los datos recibidos a través de MQTT brinda flexibilidad para identificar patrones anómalos y responder de manera inmediata. Asimismo, la implementación de un bucle de control que verifica constantemente la conexión y la calidad de los datos asegura que el sistema mantenga su operatividad sin intervención manual. Esto convierte a *ShieldNet* en una solución robusta y escalable para la detección temprana de irregularidades en entornos críticos.
- En este proyecto se aprendió a combinar Programación Orientada a Objetos (POO) con el uso de sensores y comunicación IoT, específicamente con un ESP32 y MQTT. Se logró procesar los datos del sensor DHT11 en tiempo real, detectar anomalías mediante una gráfica de control basada en Six Sigma y visualizar la información en una interfaz gráfica. Además, se implementó un sistema para almacenar solo los valores críticos en un archivo JSON, lo que optimiza el almacenamiento y facilita el análisis de datos relevantes.
- La implementación de este sistema permitió entender mejor cómo automatizar la detección de anomalías en sensores de temperatura. Gracias a la integración de

MQTT, podemos monitorear los datos de forma remota y en tiempo real, lo que demuestra la importancia de este tipo de tecnologías para aplicaciones de monitoreo y control en IoT. Además, la experiencia adquirida con el manejo de archivos, la visualización de datos y el análisis estadístico refuerza la utilidad de Python para desarrollar soluciones inteligentes y eficientes.

Referencias

IBM (s.f), ¿Qué es la industria 4.0? Recuperado de <https://www.ibm.com/es-es/topics/industry-4-0>

PTC (s.f), ¿Qué es la industria 4.0 y cómo funciona? Recuperado de <https://www.ptc.com/en/solutions/digital-manufacturing/industry-4-0>

SAP Concur Team (2021), 8 beneficios de la industria 4.0, Recuperado de <https://www.concur.co/blog/article/beneficios-industria-4.0>

Cisco (s.f), ¿What is Industry 4.0?, Recuperado de
<https://www.cisco.com/site/us/en/learn/topics/networking/what-is-industry-4-0.html>

Jordy S. & Santiago S.(2020), “INTERNET DE LAS COSAS”, Recuperado de
https://upcommons.upc.edu/bitstream/handle/2117/100921/LM08_R_ES.pdf

PTC (2025), Internet industrial de las cosas (IIoT), Recuperado de
<https://www.ptc.com/en/technologies/iiot>

IBM (2025), ¿Qué es el internet de las cosas?, Recuperado de
<https://www.ibm.com/mx-es/topics/internet-of-things>

Anna Gerber & Jim Romeo (31 de Enero del 2020), Connecting all the things in the Internet of Things, Recuperado de <https://developer.ibm.com/articles/iot-lp101-connectivity-network-protocols/>

Mario Calleja Collado, Joan Guasch Llobera (02 de mayo de 2019) Universidad Politécnica de Catalunya, “Monitorización de sensores con arduino utilizando el protocolo MQTT” , de Recuperado de
<https://upcommons.upc.edu/bitstream/handle/2117/134193/memoria.pdf?sequence=1&isAllowed=y>

Antonio León Fernández (2 de Julio de 2020), Universidad Politécnica de Valencia, “*DESARROLLO DE UNA APLICACIÓN IoT PARA EL ENVÍO DE IMÁGENES MEDIANTE EL PROTOCOLO MQTT*” , Recuperado de
<https://riunet.upv.es/bitstream/handle/10251/152408/Mahedero%20-%20Desarrollo%20de%20una%20aplicación%20IoT%20para%20el%20envío%20de%20imágenes%20mediante%20el%20protocolo%20MQTT..pdf?sequence=1&isAllowed=y>

Cameron, N. (2023). Microcontrolador ESP32. En: ESP32 Formatos y Comunicación. Serie Maker Innovations. Apress, Berkeley, CA. Recuperado de https://doi.org/10.1007/978-1-4842-9376-8_1

Gay, W. (2018). Sensor DHT11. En: Raspberry Pi avanzado. Apress, Berkeley, CA. Recuperado de https://doi.org/10.1007/978-1-4842-3948-3_22

Estado Peruano (s.f), SENAMHI, Recuperado de <https://www.gob.pe/institucion/senamhi/organizacion>

SENAMHI (s.f), “Normales climáticas estándares y medias 1991-2020”, Recuperado de <https://www.senamhi.gob.pe/?&p=normales-estaciones>

Jiménez, W. C. & González, E. S. (2021). Protoboard configurable por software, denominada “SOFTBOARD”. Recuperado de: <http://hdl.handle.net/11349/29055>

OpenAI. (2023). ChatGPT (versión del 15 de julio) [Modelo de lenguaje de gran tamaño]. <https://chatgpt.com>

Jumper. (s.f), En Wikipedia, Recuperado de [https://es.wikipedia.org/wiki/Jumper_\(inform%C3%A1tica\)#::~:~:text=En%20electr%C3%B3nica%20y%20en%20p%C3%A1rticular%20en%20inform%C3%A1tica%20un%20c%C3%ADrculo%20el%C3%A9ctrico%20del%20que%20forma%20parte%20de%20las%20conexiones.](https://es.wikipedia.org/wiki/Jumper_(inform%C3%A1tica)#::~:~:text=En%20electr%C3%B3nica%20y%20en%20p%C3%A1rticular%20en%20inform%C3%A1tica%20un%20c%C3%ADrculo%20el%C3%A9ctrico%20del%20que%20forma%20parte%20de%20las%20conexiones.)