UNIVERSIDAD CATÓLICA
DE SANTIAGO DE GUAYAQUIL

FACULTAD DE EDUCACIÓN TÉCNICA PARA EL DESARROLLO

CARRERA INGENIERÍA EN ELÉCTRICO MECÁNICA

TEMA:

Supervisión de la calidad de aire en ambiente cerrado usando tecnología embebida.

AUTOR:

Jerez Ronquillo, Diego Antonio

Trabajo de Integración Curricular previo a la obtención del título de INGENIERO ELÉCTRICO MECÁNICO CON MENCIÓN EN GESTIÓN EMPRESARIAL

TUTOR:

Ing. Bohórquez Escobar, Celso Bayardo. PhD.

Guayaquil, Ecuador

01 de septiembre del 2025

UNIVERSIDAD CATÓLICA
DE SANTIAGO DE GUAYAQUIL

FACULTAD DE EDUCACIÓN TÉCNICA PARA EL DESARROLLO

CARRERA INGENIERÍA EN ELÉCTRICO MECÁNICA

CERTIFICACIÓN

Certificamos que el presente Trabajo de Integración Curricular fue realizado en su totalidad por **Jerez Ronquillo, Diego Antonio**, como requerimiento para la obtención del título de **Ingeniero eléctrico mecánico con mención en gestión empresarial**.

TUTOR:

_____

Ing. Bohórquez Escobar, Celso Bayardo. PhD.

DIRECTOR DE CARRERA

_____

Ing. Bohórquez Escobar, Celso Bayardo. PhD.

Guayaquil, 01 de septiembre del 2025

UNIVERSIDAD CATÓLICA
DE SANTIAGO DE GUAYAQUIL

FACULTAD DE EDUCACIÓN TÉCNICA PARA EL DESARROLLO

CARRERA INGENIERÍA EN ELÉCTRICO MECÁNICA

DECLARACIÓN DE RESPONSABILIDAD

Yo, Jerez Ronquillo, Diego Antonio

DECLARO QUE:

El trabajo de Integración Curricular **Supervisión de la calidad de aire en ambiente cerrado usando tecnología embebida,** previo a la obtención del título de ingeniero eléctrico mecánico con mención en gestión empresarial, ha sido desarrollado respetando derechos intelectuales de terceros conforme las citas que constan en el documento, cuyas fuentes se incorporan en las referencias o bibliografías. Consecuentemente este trabajo es de mi total autoría.

En virtud de esta declaración, me responsabilizo del contenido, veracidad y alcance del Trabajo de Integración Curricular referido.

Guayaquil, 01 de septiembre del 2025

EL AUTOR

_____

Jerez Ronquillo, Diego Antonio

UNIVERSIDAD CATÓLICA
DE SANTIAGO DE GUAYAQUIL

FACULTAD DE EDUCACIÓN TÉCNICA PARA EL DESARROLLO

CARRERA INGENIERÍA EN ELÉCTRICO MECÁNICA

AUTORIZACIÓN

Yo, Jerez Ronquillo, Diego Antonio

Autorizo a la Universidad Católica de Santiago de Guayaquil, la publicación, en la biblioteca de la institución del Trabajo de Integración Curricular: **Supervisión de la calidad de aire en ambiente cerrado usando tecnología embebida,** cuyo contenido, ideas y criterios son de mi exclusiva responsabilidad y total autoría.

Guayaquil, 01 de septiembre del 2025

EL AUTOR

_____

Jerez Ronquillo, Diego Antonio

UNIVERSIDAD CATÓLICA
DE SANTIAGO DE GUAYAQUIL

FACULTAD DE EDUCACIÓN TÉCNICA PARA EL DESARROLLO

CARRERA INGENIERÍA EN ELÉCTRICO MECÁNICA

REPORTE DE COMPILATIO

CERTIFICADO DE ANÁLISIS
magister

Avance Final_Diego Jerez_TesisA2025[1]-2

5% Textos sospechosos

5% Similitudes
< 1% similitudes entre comillas
< 1% entre las fuentes mencionadas
42% Idiomas no reconocidos (ignorado)
2% Textos potencialmente generados por la IA (ignorado)

Nombre del documento: Avance Final_Diego Jerez_TesisA2025[1]-2.docx
ID del documento: 196baa0c9738d12d2718d42c621f6adcc72cb9bf
Tamaño del documento original: 2,89 MB

Depositante: Ronnie Alexander Bonilla Sánchez
Fecha de depósito: 26/8/2025
Tipo de carga: interface
fecha de fin de análisis: 26/8/2025

Número de palabras: 19.861
Número de caracteres: 174.644

Ubicación de las similitudes en el documento:

Fuentes principales detectadas

N°	Descripciones	Similitudes	Ubicaciones	Datos adicionales
1	Avance Final_Tesis_Ariel Angulo_B2023.docx | Avance Final_Tesis_Ariel ... 47cd3ca Viene de de mi grupo 18 fuentes similares	< 1%		Palabras idénticas: < 1% (288 palabras)
2	localhost | Estudio del desempeño de un sistema de climatización invertir alimen... http://localhost:8080/xmlui/bitstream/3317/190164/1/T-UCSG-POS-MIELE-4.pdf.txt 27 fuentes similares	< 1%		Palabras idénticas: < 1% (364 palabras)
3	TESIS PARRALES CORREGIDO.docx | TESIS PARRALES CORREGIDO 42ee091 Viene de de mi grupo 27 fuentes similares	< 1%		Palabras idénticas: < 1% (238 palabras)
4	repositorio.ucsg.edu.ec | Diseño eléctrico de subestación de 5 MVA 69/13.8 KV e... http://repositorio.ucsg.edu.ec/bitstream/3317/23462/3/UCSG-C432-22977.pdf 28 fuentes similares	< 1%		Palabras idénticas: < 1% (232 palabras)
5	localhost | Estudio y diseño en media y baja tensión del sistema eléctrico -fase 2- ... http://localhost:8080/xmlui/bitstream/3317/16220/3/T-UCSG-PRE-TEC-IIM-263.pdf.txt 10 fuentes similares	< 1%		Palabras idénticas: < 1% (173 palabras)

Fuentes con similitudes fortuitas

N°	Descripciones	Similitudes	Ubicaciones	Datos adicionales
1	Documento de otro usuario #64081c Viene de de otro grupo	< 1%		Palabras idénticas: < 1% (31 palabras)
2	repositorio.espe.edu.ec | Identificación y evaluación de riesgos financieros en la ... http://repositorio.espe.edu.ec/handle/21000/8602	< 1%		Palabras idénticas: < 1% (23 palabras)
3	doi.org | Wireless Sensor Networks for Environmental Monitoring https://doi.org/10.3390/s21041172	< 1%		Palabras idénticas: < 1% (16 palabras)
4	doi.org | A secure and energy-efficient routing using coupled ensemble selection ... https://doi.org/10.1038/s41598-024-82635-w	< 1%		Palabras idénticas: < 1% (17 palabras)
5	doi.org | Arquitectura de microservicios basada en contenedores para despliegue... https://doi.org/10.62451/rep.v3i1.73	< 1%		Palabras idénticas: < 1% (19 palabras)

Fuentes mencionadas (sin similitudes detectadas) Estas fuentes han sido citadas en el documento sin encontrar similitudes.

1 https://doi.org/10.3390/brainsci13040683
2 https://repositorio.une.edu.pe/entities/publication/repositorio.une.edu.pe
3 https://webbylab.com/blog/iot-architecture/
4 https://rdi.uncoma.edu.ar/handle/uncomaid/18734
5 https://www.researchgate.net/publication/366518443_Home_Automation_with_Arduino_IOT_Cloud

Atentamente,

Ing. Bohórquez Escobar, Celso Bayardo. PhD.

AGRADECIMIENTO

Me gustaría agradecer a todos los que han contribuido en mi vida académica y verdaderamente que culminar esta etapa se convirtió en algo muy adverso por diferente motivos, en el 2012 tuve que dejar la carrera para poder lograr los objetivos laborales y económicos que me había propuesto desde niño y conforme pasaba el tiempo cada vez que alcanzaba cumplir una meta tenía menos tiempo y pensaba que sería innecesario o imposible retomar mis estudios, aunque no tenía la obligación económica de terminarlos si tenía la obligación moral como hijo, ahora como padre y darle esto a mis mentores, es por eso que le doy las gracias a mi madre y padre por haberme dado todas las herramientas que he necesitado a lo largo de mi vida para llegar hasta estos momentos.

El tiempo de Dios es perfecto y no podía ser antes ni después agradecerle a mi compañera de vida, mi única amiga, mi brillante asistente y mi esposa amada que ha estado conmigo en todos los procesos y cambios de mi vida que junto a mi hijos son mi mayor motivación.

Quisiera agradecerles también a mis hermanos Gisel y Fabi por sus consejos desde niño que siempre me han permitido reflejarme en ellos de lo que quería para mi vida después de 8 y 12 años más.

Dios siempre ha puesto ángeles en mi camino que le han dado un plus a mi vida y en la culminación de esta etapa no faltaron, gracias a mis suegros que siempre están ahí para hacerme la vida más sencilla y feliz, gracias a todos mis profesores y tutores por su predisposición y apoyo a lo largo de todo este tiempo, gracias a mi familia de trabajo.

Jerez Ronquillo, Diego Antonio

DEDICATORIA

Dedico, esta tesis a mis padres, a mi esposa y a mis hijos que son los que me han motivado durante todo este tiempo a terminar esta etapa de mi vida y que este proyecto se haya hecho posible.

Jerez Ronquillo, Diego Antonio

UNIVERSIDAD CATÓLICA
DE SANTIAGO DE GUAYAQUIL

FACULTAD DE EDUCACIÓN TÉCNICA PARA EL DESARROLLO

CARRERA INGENIERÍA EN ELÉCTRICO MECÁNICA

TRIBUNAL DE SUSTENTACIÓN

f. _____

ING. BOHÓRQUEZ ESCOBAR, CELSO BAYARDO. PhD.

DIRECTOR DE CARRERA

f. _____

ING. UBILLA GONZÁLEZ, RICARDO XAVIER, MSC.

COORDINADOR DE ÁREA

f. _____

ECON. ERIKA PAOLA, ARZUBE MENDOZA, Mgs.

OPONENTE

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

Capítulo 2

Capítulo 3

INDICE DE TABLAS

Capítulo 3

RESUMEN

El presente trabajo de integración curricular se sitúa en el diseño de un sistema de una red de nodos sensores MQ-135 controlados por microcomputadoras ESP32, conectados a la nube de Arduino y apoyados por una aplicación en Python, dejará la monitorización en vivo de los indicadores más importantes sobre la calidad del aire en un taller automotriz de Guayaquil. El objetivo central se basa en analizar la calidad de aire en un ambiente cerrado usando tablas de datos estándares nacionales e internacionales que permitan generar alarmas de peligro, si algún parámetro se sale de las normativas. Se desarrolla un sistema eléctrico, electrónico que permita medir los parámetros para medir la calidad de aire. La metodología se hará usando un método combinado. En la etapa primera, se llevará a cabo un examen detallado de libros y reglas sobre calidad del aire, sensores MQ-135, microcontroladores IoT y espacios en la nube. Después, se creará la red de sensores: se elegirán lugares importantes dentro del taller, se ajustarán los MQ-135 con modelos de gases y se programarán los ESP32 para tener y mandar datos por medio de MQTT.

Palabras claves: ESP32, Arduino Cloud, MQTT, MQ-135, IoT, Python.

ABSTRACT

This curricular integration work is based on the design of a system of a network of MQ-135 sensor nodes controlled by ESP32 microcomputers, connected to the Arduino cloud and supported by a Python application, will allow live monitoring of the most important indicators on air quality in an automotive workshop in Guayaquil. The central objective is based on analyzing air quality in a closed environment using national and international standard data tables that allow generating danger alarms if any parameter falls outside the regulations. An electrical and electronic system is developed to measure the parameters to measure air quality. The methodology will be done using a combined method. In the first stage, a detailed examination of books and rules on air quality, MQ-135 sensors, IoT microcontrollers and cloud spaces will be conducted. Next, the sensor network will be created: important locations will be chosen within the workshop, the MQ-135s will be adjusted with gas models, and the ESP32s will be programmed to receive and send data via MQTT.

Keywords: ESP32, Arduino Cloud, MQTT, MQ-135, IoT, Python.

Capítulo 1: Descripción general del trabajo de titulación

1.1 Introducción

La calidad de aire en locales cerrados, como los talleres para arreglar carros tiene un efecto neto en la salud de los trabajadores. La aspiración constante de gases tales como el dióxido de carbono, amoniaco y compuestos orgánicos volátiles puede causar desde irritaciones en las vías respiratorias hasta enfermedades graves lo cual tiene un riesgo laboral alto. En el pasado, estos lugares se miraban con muestreos aislados con aparatos de laboratorio externo un sistema caro y poco común que hace más difícil notar picos de contaminación (Gutiérrez, 2024).

Las herramientas de la Era 4.0 dan una opción: hacer redes de sensores listos unidas a plataformas en nube por medio de protocolos IoT. En este escenario, el sensor MQ-135 y el pequeño computador ESP32, por su bajo precio y habilidades para hablar entre partes, son buenas opciones para poner un sistema que vigila sin parar. Además, la plataforma Arduino Cloud deja guardar, trabajar y ver los datos al instante sin necesitar su propio servidor (Estrada et al., 2025).

Para hacer más fácil entender las mediciones y activar alarmas cuando los niveles de cosas perjudiciales superan los límites que ponen normas nacionales e internacionales, se hará una aplicación en Python. Este programa dará una pantalla grafica para ver cambios, guardar historiales y

mandar avisos rápidos, ayudando a evitar problemas y seguir con requisitos del medio ambiente en el taller de Guayaquil.

1.2 Antecedentes

En la última década, muchas investigaciones han analizado el uso de redes inalámbricas con sensores (WSN) para el control del medio ambiente en lugares trabajo. Estos estudios muestran la efectividad de formas como en estrella y malla para cubrir y asegurar que los datos sean confiables, además del uso de programas sencillos como MQTT para enviar bien información a la nube (González, 2023).

El medidor de gas MQ-135 se ha calibrado con tipos de gases conocidos y comparado con aparatos de laboratorio. Esta prueba muestra que puede encontrar cosas malas en el aire dentro de casas. Estudios basados en reglas de la Organización Mundial de la Salud (OMS) y la Agencia de Protección Ambiental de EE. UU. (EPA) han creado forma de cambiar lo que se lee para hacer medidas mejores cuando hace frío o calor y también si hay mucha o poca humedad (Rodríguez, 2024).

Por otro lado, el pequeño cerebro ESP32 es hoy muy popular como solución ahorrativa y útil para trabajos de IoT. Sus dos partes de trabajo, conexión a Wi-Fi o Bluetooth, y varios caminos para medición de ADC hacen fácil agregar muchos sensores en un solo lugar. Varios usos han empleado estas cualidades para pasar información directo a sitios en línea, bajando lo complicado de armar el hardware (Cruz et al., 2023).

Finalmente las plataforma Arduino Cloud se ha usado en prototipos debido a que es amable y tiene herramientas como "Things", "Variables" y "Dashboards" que facilitan mucho hacer aplicaciones para ver datos. Los avances con Python, con librerías como paho-mqtt para conectar y tkinter o PyQt para el diseño visual, han mostrado su buena eficacia al manejar datos y al enviar alertas (Fernández & Ordóñez, 2021).

1.3 Definición del Problema

En el taller de arreglo de carros que se analizó en Guayaquil, la falta de un sistema que monitoree constantemente la calidad del aire dificulta saber las cantidades de gases peligrosos en tiempo real. Hoy, las mediciones se hacen de vez en cuando con equipos que vienen de fuera, lo cual no asegura que el personal está protegido al instante frente a cambios bruscos en los niveles de contaminación.

Este falta lleva a dos problemas importantes: primero, la exposición larga de los trabajadores a químicos malos sin avisos rápidos; segundo, la dificultad de hacer registros viejos claros que juntan picos altos de sustancias malas con trabajos específicos del taller. Esto hace difícil poner en marcha medidas preventivas y rendir cuentas a las autoridades ambientales.

Así que, el problema se define como la falta de una solución tecnológica combinada que deja la compra, envío, vistazo y visión en vivo de datos buenos del aires; de modo que se puedan hacer alarmas automáticas cuando se pasan los límites fijados por reglas locales e internacionales.

1.4 Justificación del Problema

La puesta en marcha de un sistema de vigilancia constante usando IoT da respuesta a la necesidad de cuidar la salud de los trabajadores, reduciendo la exposición a sustancias dañinas con avisos rápidos. Desde un punto de vista médico y laboral, esto puede bajar el número de problemas de pulmones y mejorar la vida en lugar de trabajo.

En ámbito legal, tener registro automático y seguro de las condiciones ambiente es buen para ver si me dentro los límites que pusieron Ministerio de Salud Pública del Ecuador, la OMS, EPA. Este no solo ayuda a evitar multas, sino también impulsa acciones más cuidadosas con el mundo.

También, el uso de partes baratas y de programa libre, sensor MQ-135, ESP32, Arduino Cloud y Python ayuda a que el plan sea accesible y fácil de copiar en otros talleres o industria pequeñas. Al combinar ideas de la actual Industria, la tesis mueve una forma nueva para crear tecnología fácil de usar que puede ser guía para otras uses en varios trabajos.

1.5 Objetivos del Problema de Investigación

1.5.1 Objetivo general

Analizar la calidad de aire en un ambiente cerrado usando tablas de datos estándares nacionales e internacionales que permitan generar alarmas de peligro, si algún parámetro se sale de las normativas.

1.5.2 Objetivos específicos

- Revisar la literatura para obtener los parámetros a considerar una calidad de aire.
- Desarrollar un sistema eléctrico, electrónico que permita medir los parámetros para medir la calidad de aire.
- Generar un software de control que permita el monitoreo y generación de alarmas en caso de que los parámetros se salgan de los estándares.

1.6 Hipótesis

La adopción de un sistema de una red de nodos sensores MQ-135 controlados por microcomputadoras ESP32, conectados a la nube de Arduino y apoyados por una aplicación en Python, dejará la monitorización en vivo de los indicadores más importantes sobre la calidad del aire en un taller automotriz de Guayaquil. Se espera que este sistema dará avisos o alertas al sobrepasar los umbrales de las normativas, mejorando el cuidado de la salud en el trabajo y asegurando que cumpla con reglas ambientales.

1.7 Metodología de Investigación

La investigación se hará usando un método combinado. En la etapa primera, se llevará a cabo un examen detallado de libros y reglas sobre calidad del aire, sensores MQ-135, microcontroladores IoT y espacios en la nube. Después, se creará la red de sensores: se elegirán lugares importantes dentro del taller, se ajustarán los MQ-135 con modelos de gases y se programarán los ESP32 para tener y mandar datos por medio de MQTT.

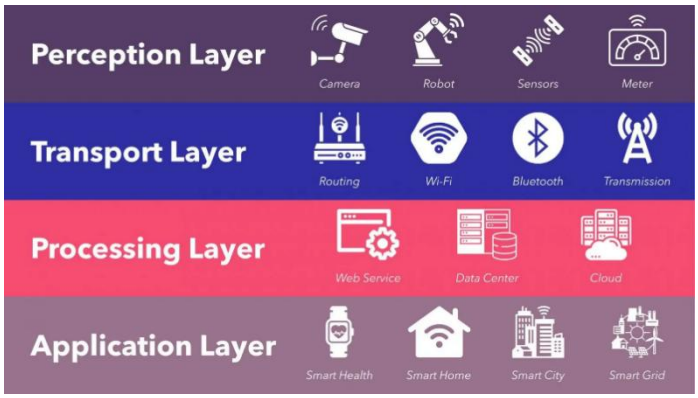
En paralelo, se hará la aplicación en Python. Primero se decidirá el diseño cliente-servidor y las llamadas a la API de Arduino Cloud; luego se hará la pantalla gráfica y se pondrá el sistema de alertas según límites. Al final, el sistema se usará en campo y se verificará comparando las lecturas con equipos de referencia.

Capítulo 2: Fundamentación Teórica

2.1 Introducción al Internet de las Cosas (IoT)

En la actualidad, hay un número creciente de objetos que se conectan al internet y pueden comunicarse entre sí como se observa en la figura 2.1. Ese conjunto de objetos se llama Internet de las Cosas (IoT). Hoy en día, hay muchos más aparatos conectados que personas utilizando el Internet. El IoT cambia la forma en que vivimos y hace cosas más fáciles. En el entorno digital actual, más artilugios se vinculan al ciber espacio y pueden hablar entre ellos. Esta grupa mezcla cosas se nombra Internet de las Cosas (IoT). Ahora hay un buen número de máquinas conectadas que gente usando ciber espacio. El IoT cambia cómo vivimos y hace cosas más simples (Vasoya, 2023).

Figura 2.1: Características que conforman los dispositivos IoT.



Nota: Etapas principales de la estructura IoT. Fuente: (Simmons, 2022)

El Internet de las Cosas (IoT, por sus nombres en inglés) se dice que es un grupo de aparatos que se conectan y pueden recoger y enviar información por internet sin que una persona tenga que hacer nada. Estos objetos, como electrodomésticos y cámaras de seguridad, o sensores industriales y autos, tienen chips, sensores y programas que les hacen

interactuar con su alrededor. Así, el IoT es la mejora hacia una conexión que está en todas partes donde los cosas normales tienen nuevas habilidades por combinar tecnología y comunicación en línea (Ñaupa, 2022).

La relevancia del IoT está en su habilidad para cambiar formas de hacer cosas, mejorar lo bien que funcionan y hacer nuevos modelos de negocio en muchos campos. Algunas de las mejores formas en que se usan son :

- Usar aparatos automáticos en casa como termostatos y luces que se pueden controlar por voz.
- Chequear salud con dispositivos usables como relojes y sensores biomédicos.
- Ver y manejar cosas grandes en la ciudad como semáforos inteligentes y manejo de basura.
- Hacer mejor la producción en fábricas con sensores que encuentran problemas o necesidades de arreglos.

Estas no solo dejan una mejor gestión de recursos, pero también abren la puerta a mejoras en áreas como la agricultura, el comercio y el transporte aumentando su importancia en la economía y vida diaria.

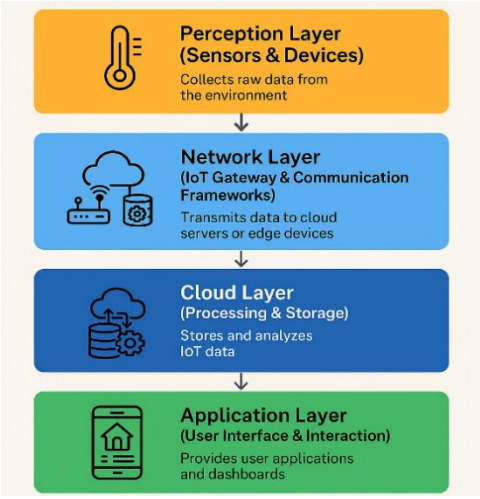
La evolución histórica del IoT empezó con los primeros intentos con aparatos conectados en los años 1980 y 1990, si bien la palabra fue usada por primera vez en 1999. Al principio, la conexión era solo para algunos artefactos, pero a medida que Internet se hizo más común y sensores y microprocesadores bajaron de precio el concepto fue ganando fuerza y tamaño. Hoy, el IoT pasó de ser una idea futura a ser una cosa real con

muchos aparatos en todo el mundo uniendo avances tecnológicos como el Edge computing y la inteligencia artificial para mejorar mucho su efecto y uso (Rossi, 2025).

2.1.1 Características principales del IoT

El Internet de las Cosas se basa en los objetos conectados a la red como se aprecia en la figura 2.2. Estos tienen sensores que recogen datos y los envían a un centro. Puede cambiar cómo se usan las cosas, por ejemplo, una llave que abre puertas sin tocarlas. Estos aparatos también pueden hablar entre sí, ayudándose mutuamente o a ti. En lugar de esperar que algo se rompa, estos cacharros pueden chequearse ellos mismos para ver si hay problemas. La parte importante del IoT es que puede cambiar la manera en que vivimos y trabajamos (Peralta, 2025).

Figura 2.2: Elementos esenciales de la estructura IoT.



Nota: Diagrama de arquitectura del IoT. Fuente: (IoTDunia, 2025)

Una de las cosas más importantes del Internet de las Cosas (IoT) es la conexión en tiempo presente, la que deja que los aparatos y sensores

cambien datos a menudo por redes fuertes y seguras. Esta conexión sirve para cuidar procesos todo el tiempo, revisar rápido la información y accionar en el momento ante sucesos vistos, lo que es clave para usos como ciudades inteligentes, salud digital o automatización industrial. También, enviar en tiempo real asegura una mezcla sin problemas entre lo físico y lo numérico dejando a los sistemas responder bien y anticiparse a necesidades del entorno que lo rodea de manera general (S. Barreto & Sandoval, 2025).

La máquina que hace cosas por sí sola y el manejo a distancia son otro parte muy importante del IOT, porque dejan manejar y usar dispositivos sin que un humano tenga que estar ahí siempre. Por la tecnología IOT, se pueden crear rutinas, cambiar parámetros y ver lo que hacen los equipos desde cualquier lugar con App del celular o sitios web, esto trae a favor como (Carrillo & Rojas, 2024):

- Mejor uso de los recursos y menor gasto en trabajar.
- Más seguridad porque se hallan fallos pronto.
- Mejor disfrute del usuario a la hora de hacerlos servicios por son únicos y respuestas. Por lo que la máquina y el manejo desde lejos suben el buen uso y la toma de decisiones basada en información real al instante.

Por último, la escalabilidad y flexibilidad son elementos importantes que hacen diferente al IoT de otras tecnologías viejas. Los sistemas IoT están hechos para agregar nuevos aparatos con facilidad y crecer cuando crecen las necesidades de los usuarios o las compañías. Esta capacidad de cambio

se consigue por estructuras que se pueden mover y formas de hablar entre aparatos que son comunes, que ayudan a unir distintas soluciones y la conexión ente varios fabricantes. Así, el IoT puede mejorar rápido para contestar a los cambias en un ambiente tecnológico o de negocio, asegurando una inversión sostenible en futuro (Decimavilla & Marcillo, 2025).

2.1.2 Arquitectura general del IoT

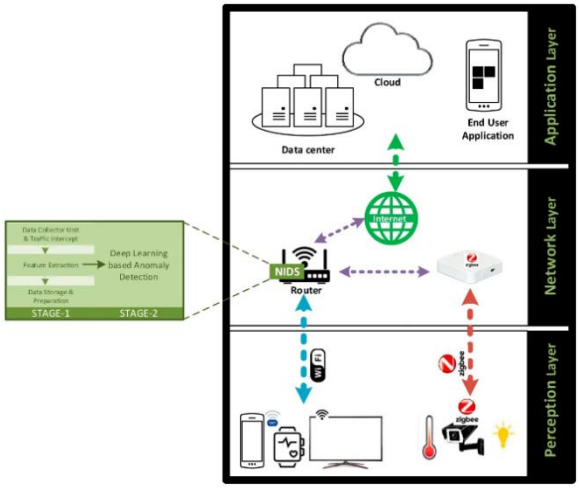
La arquitectura del Internet de las Cosas (IoT) tiene varios partes importantes que permiten conectar y manejar bien dispositivos inteligentes como se observa en la figura 2.3. Los partes básicos incluyen los sensores y actuadores que recogen datos del entorno o hacen cosas como mover; los dispositivos de cálculo con análisis filtración; y modelos para comunicarse entre aparatos y plataformas principales. También hay redes que ayudan a enviar la información, plataformas en nube para guardar o calcular datos complejos, y apps que mostraron o controlan los gadgets conectados a la red de forma respectiva (López & Isabel, 2024).

Figura 2.3: Estructura de las capas principales del IoT.

Nota: Etapas básicas fundamentales del cifrado IoT. Fuente: (Oliynyk, 2023)

Cada uno de estos componentes cumple una tarea clave en el sistema de la IoT. Los sensores y los actuadores son los encargados de conectar con el mundo real, ayudando a recoger datos útiles o hacer trabajos exactos. Por otro lado, los módulos para el procesamiento local pueden limpiar o ver datos antes de enviarlos a la nube así ahorrando ancho de vándalos aparatos de comunicación se aseguran de que los datos viajen bien y seguros mientras que las plataformas en la nube permite el almacenamiento grande y un análisis avanzado como inteligencia artificial o predicciones como se observa en la figura 2.4 (Rey, 2023).

Figura 2.4: Ejemplo de detección de anomalías mediante IoT.



Nota: Uso de redes neuronales profundas en IoT. Fuente: (Dilmegani, 2025)

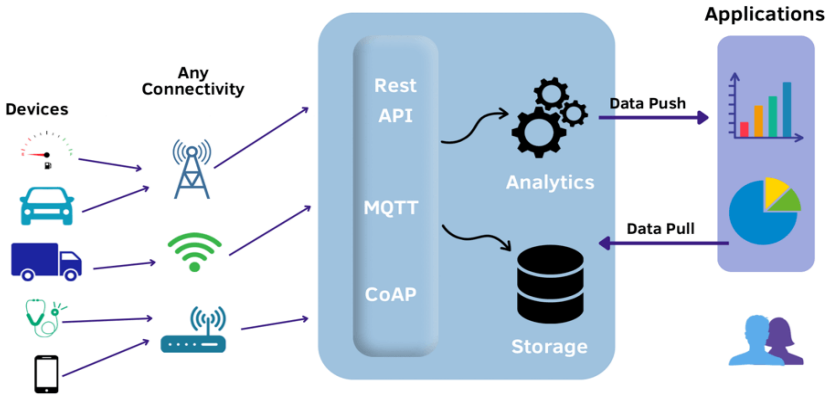
La conexión entre estas partes es muy importante para el buen andar del IoT, ya que cada uno necesita del otro para tener un trabajo seguido y ordenado. Por ejemplo, los sensores hacen datos que tienen que pasar a través de la red a los sistemas de tratamiento y almacenaje en la nube, donde las aplicaciones pueden llegar y usar la información para hacer acciones automáticas o mostrar cosas a alguien que la usa. Esta unión fuerte

de elementos deja tener cosas como automación, control lejano y tamaño adaptable, lo que hace que el sistema IoT cambie a distintos lugares y cantidades de aparatos que están unidos entre sí (Rovira, 2023).

2.1.3 Nivel de capa de dispositivo de la tecnología IoT

En esta capa los sensores y motores son muy importantes porque ayudan a que interaccionen el mundo real y el sistema digital como se aprecia en la figura 2.5. Los sensores cogen datos útiles del entorno como la temperatura, humedad, luz, movimientos o presencia de gases y los motores hacen cosas físicas en respuesta al dar señales como abrir válvulas, encender motores o cambiar luces. Esta recogida y acción de datos es la base para que los aparatos en la red puedan ver y cambiar el mundo físico sin ayuda (Torres et al., 2024).

Figura 2.5: Interacción de dispositivos, sensores redes y recursos con IoT.



Nota: Estructuración de los distintos elementos que componen la nube IoT.
Fuente: (Sayanekar, 2024)

La tarea más importante de los dispositivos en esta capa es doble: por un lado, recoger datos del entorno mediante sensores; y por otro, hacer cosas importantes con actuadores siguiendo indicaciones de otras capas del

sistema. Así, la capa de dispositivo sirve como puente entre el mundo físico y el digital permitiendo que el sistema IoT tenga decisiones fundamentadas y realice actos precisos. También, estos dispositivos a menudo son construidos para trabajar solos y eficientemente garantizando la continua transmisión de datos relevantes para el funcionamiento global de la red (Herrera et al., 2021).

Algunos ejemplos de aparatos en la capa de dispositivo del IoT son (Kirvan & Gillis, 2025):

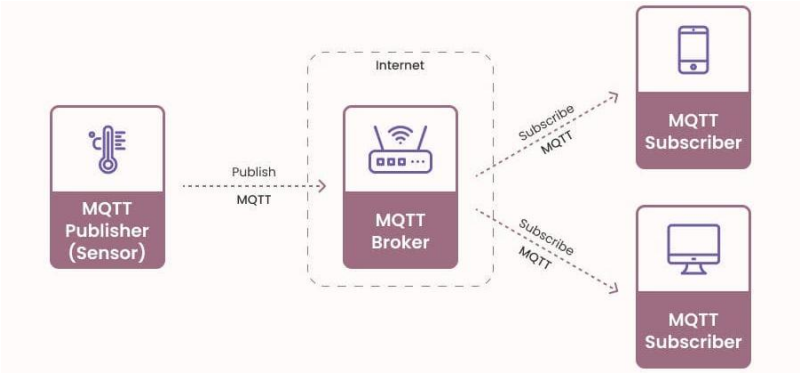
- Medidores de calor y humedad que se usan en la agricultura precisa.
- Cámaras inteligentes para vigilar la seguridad en casa y negocios.
- Aparatos para ver ritmo cardíaco o movimiento en la salud.
- Actuadores en sistemas del hogar, como cerraduras inteligentes térmicas. Estos aparatos, por su habilidad para unirse a redes y compartir información, son la base de muchas apps nuevas dentro del mundo IoT.

2.1.4 Nivel de capa de red de la tecnología IoT

La capa de red en la arquitectura de IoT tiene un papel muy importante al manejar los protocolos de comunicación que dejan actuar entre equipos, puntos de acceso y servidores. Estos protocolos, como se aprecia en la figura 2.6 el MQTT, Copa y HTTP, se eligen según la eficiencia, el gasto de energía y las necesidades de enviar datos de cada aplicación. Por ejemplo MQTT es muy común porque usa poco ancho de banda y puede funcionar en áreas con poco recursos mientras HTTP podría ser mejor para aplicaciones donde es importante trabajar juntamente con servicios web clásicos. Así la elección

correcta del protocolo define lo bien que funciona y lo fiable que es la comunicación en entornos de IoT, haciendo fácil la unión entre dispositivos diferentes (Tran et al., 2024).

Figura 2.6: Protocolo de comunicación empleado en MQTT.



Nota: Elementos que integran el protocolo MQTT. Fuente: (Rupareliya, 2025)

Las tecnologías de red usadas en la capa de red de IoT van desde Conectores cableados normales hasta opciones inalámbricas modernas hechos solo para dispositivos listos. Entre las más usadas están Wi-Fi, Sigue, Bluetooth Low Energy (BLE) y soluciones de largo alcance como LoRaWAN y NB-IoT. Cada una de estas técnicas trae buenas cosas diferentes, como poca energía o un alcance grande o fácil unión con sistemas ya hechos. Por ejemplo, LoRaWAN es bueno para tareas que necesitan cubrir toda una ciudad con aparatos de baja de; mientras que Wi-Fi es mejor para espacios donde hay mucha demanda por ancho. La mezcla sabia de estas técnicas deja que las redes IoT se ajusten a varios situaciones y requisitos operacionales (Liu et al., 2022).

La seguridad en la en de datos es un punto muy importante en la capa de red, ya que los aparatos IoT a menudo manejan información sensible y

trabaja en lugares que pueden ser atacados. Para asegurar que los datos no cambien ni se vean por extraños, se usan métodos como cifrado de extremo a extremo, chequeo mutuo entre aparatos y el uso de redes privadas virtuales (VPN). También es importante manejar certificados digitales y credenciales seguras para evitar que personas no autorizadas intenten entrar. Ya que una debilidad en esta capa puede dañar todo el ecosistema de IoT, la fortaleza de las medidas de seguridad que se ponen es un gran factor para el éxito y confianza de las apps IoT (Mazhar et al., 2023).

2.1.5 Nivel de capa de nube de la tecnología IoT

La capa de nube en la arquitectura de IoT hace un trabajo importante al guardar y procesar los datos que crean los aparatos conectados. Cuando los sensores o aparatos IoT juntan datos, estos van a la nube donde se pueden analizar con buenas herramientas. Este proceso deja que grandes cantidades de datos sean manejadas bien ayudando a sacar información útil y dejando la toma de decisiones inteligentes. También la nube ofrece escalabilidad, lo que permite que el sistema un número cada vez mayor de aparatos y datos sin perder velocidad (Ficili et al., 2025).

Los servicios en la nove para IoT están hechos especialmente para ayudar a la variedad y delicadeza de las cosas y usos del mundo IoT. Estos servicios tienen plata formas que unen muchos tipos de sensores, herramientas para leer bien lo que se junta, y formas de manejar que hacen fácil ver y cambiar los aparatos desde lejos. Con estas habilidades las empresas pueden crear soluciones IoT más rápido y bien; ganando por la

libertad y tener tecnologías nuevas sin usar dinero en cosas que ya tienen (Hasan, 2022).

El usar la nube en IoT trae muchas cosas buenas que ayudan a la función y el alcance de las maneras de hacer. Entre los grandes puntos buenos están (Zhao et al., 2024):

- Bajada de dinero al quitar necesidad de usar servidores reales.
- Cobertura que puede cambiar para unir a montones de datos o aparatos nuevos.
- Entrada universal y está disponible todo el tiempo la información y servicios.
- Subida en la seguridad con formas nuevas de curar datos.

2.1.6 Nivel de capa de aplicación de la tecnología IoT

En el grupo de aplicación del Internet de las Cosas (IoT), las interfaces de usuario juegan un papel muy importante al dar la unión entre los personas que usan el sistema y los dispositivos inteligentes. Estas interfaces pueden estar en formas como apps para móviles, paneles web o ayudantes de voz; permitiendo a los individuos observar, manejar y organizar sus aparatos de manera sencilla. El sentir del usuario es un punto clave aquí, ya que qué fácil es usar y acceder determina mucho el éxito de una solución IoT. Así, un diseño bueno de la interfaz ayuda a tomar la técnica y mejora la felicidad de quien usa el sistema, lo cual es muy importante tanto en casas como en fábricas (I. Ullah et al., 2025).

El estudio y la muestra de datos son otro parte importante de la capa de programa en la estructura del IoT. Al recoger y cambiar los datos en las partes bajas, esta información se enseña a los usuarios de forma clara y útil. Las plataformas del IoT a menudo traen herramientas para analizar y paneles que ayudan a ver patrones, tendencias y posibles fallos en el usar de los aparatos. Esto no solo ayuda a elegir mejor, sino que también suma a mejorar procesos y hallar fallos rápidos (Svalina et al., 2021).

La persona libación y la automatización son cosas importantes en la parte de la aplicación porque dejan cambiar el trabajo de los aparatos IoT para adaptarse a lo que la gente quiere y necesita. Usando normas que crea el usuario o por medio del aprendizaje mecánico se puede hacer respuestas sólidas a ciertas acciones, mejorando así usar mejor los recursos y hacer más eficaz el trabajo. Entre los mejores efectos que tiene esta capacidad están (Rajkumar et al., 2023):

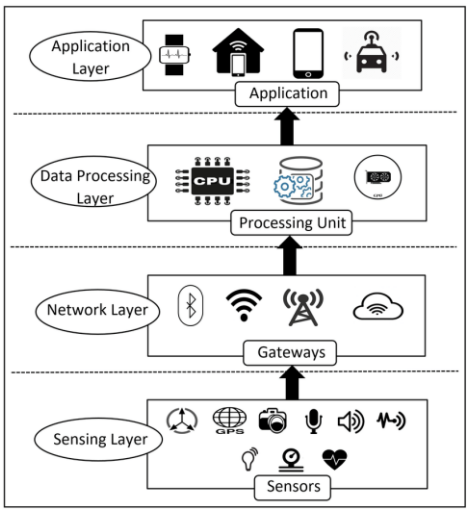
- Cambiar solo parámetros en dispositivos, según horarios o situaciones del entorno.
- Novedades basadas sobre lo que hace cada persona.
- Hacer automáticamente tareas diarias, así subiendo la comodidad y seguridad.

2.2 Introducción general de las redes de sensores inalámbricos en entornos industriales

Las redes de sensores inalámbricos (WSN) están hechas de muchos aparatos pequeños, de un reducido precio y bajo uso de energía, llamados

nodos que pueden sentir, hacer cálculos y enviar de manera inalámbrica datos del entorno a centros de procesamiento u otros nodos en la red. Las tareas principales de las WSN es recoger información física o del ambiente, como calor, presión o movimiento como se observa en la figura ; luego hacer un procesamiento local y la forma eficaz de enviar esta información por la red para su análisis o acciones. Estas redes usan mejoras en tecnología para sentir, comunicaciones sin hilos y computación integrada; funcionan por sí solas en espacios difícil o lejanos dando datos al momento (Priyadarshi, 2025).

Figura 2.7: Interacción de la capa de aplicación con las demás capas IoT.



Nota: Etapas de interacción de los sensores con la capa de aplicación de IoT. Fuente: (Sikder et al., 2020)

La adopción de redes de sensores inalámbricos en entornos industriales ofrece varias ventajas significativas, sobre todo por su flexibilidad, escalabilidad y buen costo. Al no necesitar mucho cableado, las WSN bajan los gastos de instalar y cuidar, mientras dejan una forma rápida deponer y cambiar los sistemas de chequeo. También, las WSN mejoran la eficiencia

al mirar constantemente los equipos a distancia, encontrar problemas temprano y llevar mejor los procesos. Así, las industrias tienen más seguridad, usan mejor los recursos y sigue bien las reglas, lo que hace a estas redes una solución atractiva para sistemas modernas de automatización y control industrial (Segal, 2024).

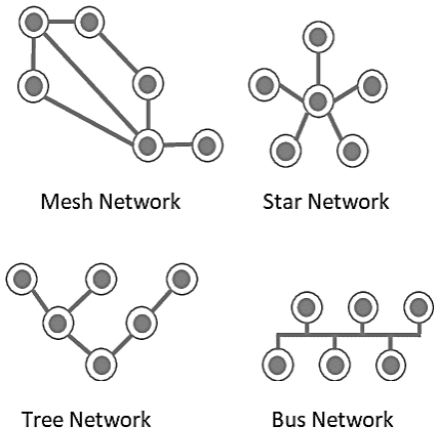
Los lugares normales de uso de redes inalámbricas en fábricas son muchos, lo que enseña la gran variedad de usos que tienen estas redes. Por ejemplo, las redes inalámbricas se usan a menudo para ver cómo están los equipos en fábricas, seguir las condiciones del clima en lugares de almacenamiento y asegurar la seguridad en lugares inseguros como plantas donde hay químicos. También se usan para el manejo de procesos en servicios públicos como plantas donde limpian el agua, la gestión de lo que hay en almacenes en tiempo real y el seguimiento del estado de estructuras en obras grandes (Lanzolla & Spadavecchia, 2021).

2.2.1 Topologías de comunicación comunes en las WSN

La topología en estrella es un tipo de estructura para hablar entre computadoras que es muy simple. Se usa en redes de sensores sin cable (WSN), sobre todo en lugares donde la simplicidad y el control principal son importantes. En esta forma, todos los nodos de sensores hablan directamente con un nodo principal o base, que es el centro para juntar y manejar datos. Esta manera trae buenas cosas: baja espera y facilidad para manejar la red; el centro maneja todo sobre ruta y coordinación. Los usos comunes de la forma estrella en industria son sistemas pequeños de

vigilancia, siguiendo bienes en zonas limitadas, y aplicaciones donde la confiabilidad depende de las fuertes habilidades de comunicación del nodo principal. Pero la mayor desventaja de la red en estrella es lo fácil que se rompe: si el nodo del medio no funciona, toda la red para, lo que la hace menos buena para usos grandes o importantes (Almarri et al., 2025).

Figura 2.8: Tipos de conexiones más usadas en WSN.



Nota: Distintos modos de transferencias de datos Y comunicación empleados en WSN. Fuente: (Shethi, 2024)

La red en malla, por otro lado, está hecha para usarse en redes de sensores que no usan cables y son más difíciles y fuertes, sobre todo en ambientes grandes y difíciles. En una red en malla, cada nodo puede hablar con varios de los nodos cercanos, lo que deja pasar los datos por diferentes caminos hasta llegar a su destino. Este armare ofrece beneficios grandes como una confiabilidad más alta y mayor capacidad para crecer o expandirse ya que esto se puede autor reparar moviendo los datos si un nodo o conexión falla. Las formas en malla son muy buenas para fábricas grandes o lugares con muchos obstáculos físicos y problemas ya que conservan la conexión

aunque no se pueda hablar directamente con la estación base. Pero lo complicado que son los protocolos de la ruta y el mayor uso de energía por la frecuente reemisión de datos dan retos; esto puede acortar la vida de batería de los nodos sensores (Ambareesh et al., 2025).

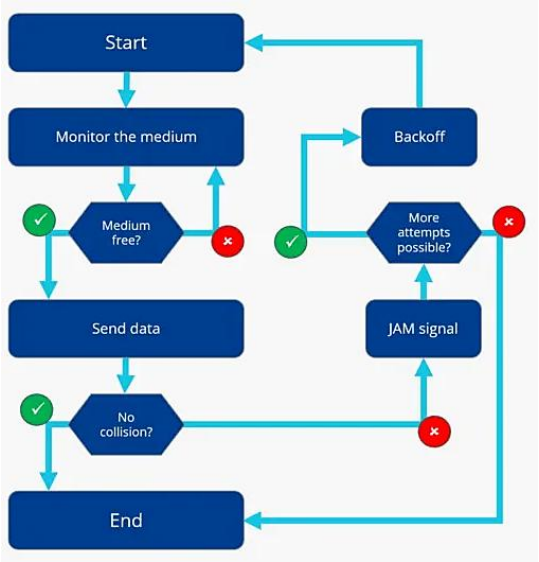
Las maneras híbridas juntan cosas de configuraciones en estrella y malla para usar lo mejor de cada una mientras bajan sus debilidades respectivas. Mucho en implementaciones de industria se usa un método mixto donde grupos de nodos hablan como estrellas con un jefe aquí, estos jefes se enlazan por red malla. Esta estructura deja controlar áreas específicas y recolectar información bien dentro de los grupos, mientras que la red principal en forma de red asegura una comunicación fuerte y que puede crecer por toda la instalación. Estos modos mezclados son normales en lugares industriales complicados, como fábricas inteligentes y sistemas que chequea energías grandes donde la confianza y poder escalar son muy necesarios. Al juntar planes topológicos, las redes híbridas pueden solucionar variados requisitos de funcionar y cambiarse a las necesidades industriales (García et al., 2025).

2.2.2 Protocolos empleados en las redes WSN de tipo industriales

En el centro del buen trabajo de las redes de sensores inalámbricos (WSN) en lugares industriales hay los protocolos de la capa que maneja el acceso al medio. Estos protocolos organizan cómo los nodos de sensores usan el espacio inalámbrico y planifican sus envíos para tener menos problemas y usar menos energía. Estos protocolos son muy

importantes para tener una buena comunicación, especialmente en lugares donde muchos aparatos pueden tratar de enviar datos al mismo tiempo. Los protocolos MAC como TDMA (Acceso Múltiple por División de Tiempo) y CSMA/CA (Acceso Múltiple por Detección de Portadora con Prevención de Colisiones) como se observa en la figura 2.9 son bastante usados y dan maneras para planificar envíos y bajar las interferencias entre nodos. En lugares de trabajo, los modos de hacer que la capa MAC funcione deben pensar en cosas malas del ambiente y ruidos que hacen las máquinas, lo que pide métodos fuertes y que cambien. El buen funcionamiento de la red depende mucho de los modos de esta capa (Sadeq et al., 2022).

Figura 2.9: Gestión de colisión en protocolo CSMA/CD.



Nota: Interacción de los datos en CSMA/CD. Fuente: (Ionos, 2020)

Los modos en la capa de red son clave para cuan grande puede ser las redes industriales sin cables, dejando el paso rápido de datos desde los censores hasta las puertas enlace o bases; incluso cuando la red va creciendo. Esos modos deben solucionar cosas como cambios que pasan

rápido, fallos y la necesidad de hablar entre muchos saltos. Modos como AODV (Vector de Distancia Ad-Demanda) y DSR (Enrutamiento Dinámico de Fuentes) son comunes por ser capaces de encontrar y guardar rutas dinámicas, al mismo tiempo que guardan energía (Wang et al., 2024).

Las características principales de los protocolos del estrato de red en las WSN industriales son (Ullah et al., 2024):

- Capacidad para adaptarse a cambios en la red.
- Manejo eficiente de nodos que cambian o se mueven.
- Al asegurar que datos lleguen bien y mejorar rutas, los protocolos del estrato de red ayudan a las WSN cumplir con los pedidos de aplicaciones para mirar y manejar industrias.

En la capa de aplicación, los protocolos hechos para recoger y manejar datos en industria juegan un papel muy importante para asegurar que la información recolectada por los sensores se entregue en un modo útil para su revisión y estudio. Estos protocolos suelen tener cosas como la unión de datos, detección de sucesos y unión con sistemas de automatización industrial. Algunos tipos comunes son MQTT (Mensajes en Cola para Telemetría) y Copa (Protocolo para Aplicación Poco Pesada), hechos para una comunicación ligera y segura en redes con pocos recursos. La capa de aplicación hace más fácil unirme los datos del WSN en las infraestructuras industriales del presente lo que ayuda a usos como el mantenimiento predictivo, control en tiempo real y mejora de procesos (Karmal, 2025).

2.2.3 Factores ambientales que afectan en tema de rendimiento de las redes WSN

Las barreras físicas, como muros, máquinas y grandes partes metálicas, son un problema difícil en luz lugares de trabajo; puede hacer que la señal sea más débil o cortar el área de conexión de las redes inalámbricas de sensores (WSN). Cuando alguien halla tales barreras las señales débiles aumentan la pérdida de datos y retrasos en la conexión. En lugares de trabajo este tema se vuelve más agresivo porque hay muchos equipos juntos y hay estanterías o cajas para guardar cosas. Para solucionar estos problemas, es importante tener en cuenta cómo son las instalaciones al hacer la red. Así se pueden poner estratégicamente los nodos de sensores para disminuir el efecto de las barreras físicas y seguir teniendo una buena trasmisión de datos (Duobiene et al., 2024).

La interferencia electromagnética (EMI) desde la maquinaria en el área laboral, se convierte en un tema muy importante que podría dañar la capacidad de las WSN. Maquinas como motores soldadoras o cambiadores de velocidad crean campos electromagnéticos que pueden distraer a la comunicación sin cables al añadir ruido o dañar datos. Esta interrupción puede conducir a más envíos, mayor consumo de energía e incluso pérdida de conexión en casos severos. Para evitar la EMI, usa un cuidado al elegir canales, esconde partes y usa métodos para comunicarte que aguanten niveles grandes de ruido (Versa, 2025).

Los cambios de calor y humedad en lugares de trabajo pueden cambiar mucho la fe y vida útil de los aparatos que miden cosas en redes WSN. Las temperaturas muy altas o bajas pueden afectar cómo funcionan las piezas electrónicas causando cambios en las mediciones de los sensores o si son graves fallas del sistema. De la misma manera, mucha humedad puede hacer que el agua se junte y corroa afectando mal a los circuitos. Para asegurar buen trabajo las WSN en lugares industriales usan sensores y cajas diseñadas para condiciones malas, así como calendarios regulares de ajuste fino y cuidado (Arman, 2022).

2.2.4 Desafíos de escalabilidad en las redes WSN de tipo industriales

La expansión de la red en ámbitos industriales trae grandes problemas para las redes de sensores sin cables (WSN), sobre todo cuando crece el número de nodos sensores para cubrir áreas más amplias o necesidades de control más difíciles. A medida que sube la cantidad de nodos, pueden aparecer fallos como la mezcla de rangos de comunicación, mayor ruido y un alto porcentaje de choques lloque baja el trabajo y fiabilidad del sistema. También las redes llenas complican manejar el gasto de energía y unir datos lo que hace difícil sostener el rendimiento que se quiere. En lugares donde hay trabajo industrial, donde una rápida expansión puede ser pedida por cambios en la forma de hacer cosas o lo que se necesita ver, estas preocupaciones sobre si se puede hacer más grande lo hacen más fuerte, lo que demuestra que hay que tener buenas ideas para diseñar redes que dejen crecer sin bajar la calidad de la comunicación (Pathak & Yadav, 2024).

El envío y cómo llegan los mensajes en grandes ajustes de WSN suman un grado extra de complicación, porque dar nombres únicos y asegurar qua los datos lleguen bien es más difícil con miles de puntos esparcidos. Los métodos de envío necesitan ser grandes y cambiar con las formas que mueven para garantizar un pase eficiente de datos a través muchos saltos. En varios casos, se usan formas de enrutamiento en niveles o centradas en datos para bajar la carga y mejorar la ampliación; pero estos modos deben equilibrar bien la usanza de los recursos de la red con el envío rápido de datos. El reto se pone más difícil en lugares de trabajo donde la estructura de la red puede cambiar mucho debido al movimiento de machinas o cosas del entorno, lo que necesita formas de enviar mensajes que se amoldan rápido para mantener la unión y bajar la perdida de paquetes (Gowda & Anandaraj, 2025).

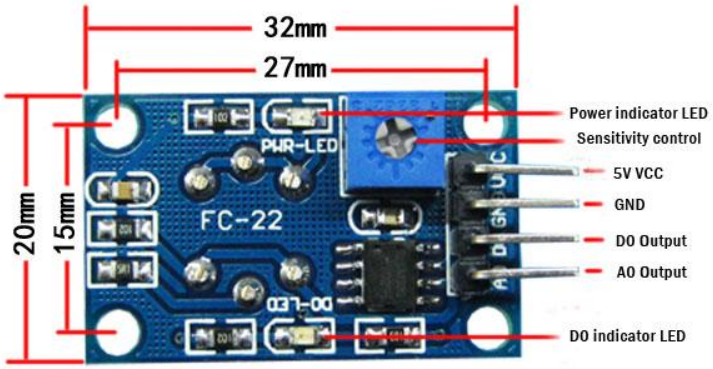
El balanceo de carga y la gestión de la congestión son muy importantes para mantener el rendimiento de la WSN cuando crece el tamaño de la red, sobre todo en entornos industriales con mucho tráfico de datos y patrones variables. Si no se usa una buena manera de distribuir cargas, algunos nodos pueden estar sobrecargados y acabar sus reservas de energía más rápido creando cuellos que empobrecen la red. Maneras como circular el trabajo adaptativo, escoger rutas dinámicamente y protocolos que notan el tráfico pueden ayudar a bajar estos problemas haciendo que el trabajo comunicativo se propague mejor en la red. Cuidar un manejo justo de los recursos y parar la congestión son muy importantes para un funcionamiento fiable en redes de sensores inalámbricos (WSN) industriales grandes donde

la vigilancia constante y el envío rápido de datos son principales (Khagga et al., 2025).

2.3 Introducción general del sensor de calidad del aire MQ-135

El sensor de calidad del aire MQ-135 como se observa en la figura 2.10 se diferencia por su diseño fuerte y sus características útiles, lo que lo hace una buena opción para encontrar muchos tipos de contaminantes en el aire. En medio del sensor hay un tubito de cerámica hecho de óxido de aluminio (Al_2O_3), cubierto con una capa que se basa en dióxido de estaño (SnO_2) y sostenido por una red de metal inoxidable para más protección y duración. Esta forma especial deja al MQ-135 reaccionar bien con compuestos gaseosos, cambiando su resistencia eléctrica según cuantos gases han estado cerca. El sensor tiene salidas simples y números, lo que deja a las personas saber datos al momento y ajustar límites para ciertas cantidades de gases. Su fácil unión y trabajo sin errores lo hacen perfecto para muchas tareas donde la revisión de la calidad del aire es muy importante. Las formas comunes en que se usa el sensor MQ-135 cubren un rango amplio, sobre todo para revisar el entorno y medir la calidad del aire en casa. Este sensor a menudo se emplea en aparatos hechos para medir el aire en casas, oficinas y lugares trabajo, ya que puede encontrar gases dañinos como amoníaco óxidos de nitrógeno, benceno humo y dióxido carbono. Además, el MQ-135 juega un rol clave en lugares donde se mide la calidad del aire que utilizan internet de cosas, sensores electrónicos y sistemas alarma; aquí la detección exacta y respuesta rápida a altos niveles de gases son muy importantes (Hassan et al., 2024).

Figura 2.10: Dimensiones y conexiones del sensor MQ-135.



Nota: Pines del sensor de calidad de aire. Fuente: (Vistronica, 2021)

Una de las ventajas grandes del MQ-135 frente a otros sensores de gas es su muy buena sensibilidad y precisión para varios gases sucios. A diferencia de sensores que solo Encuentra un tipo de gas, el MQ-135 es bueno para identificar varios contaminantes, como dióxido de carbono, amoniaco, benceno y óxidos de nitrógeno en un mismo dispositivo. Esta amplia habilidad para detectar se muestra en varias ventajas prácticas (Chodorek et al., 2022):

- Costo bajo porque un sensor puede reemplazar la necesidad de muchos detectores especiales.
- Diseñar circuitos e integrar sistemas es fácil.
- Seguimiento mejorado del ambiente al dar datos completos sobre la calidad del aire.
- También, su rápido tiempo de reacción y su calibración sencilla refuerzan su nombre como una solución confiable para el seguimiento continuo de la pureza del aire en lugares dinámicos.

2.3.1 Materiales y construcción del sensor

El sensor MQ-135 trabaja usando un método de detección quimio resistivo, que quiere decir que su resistencia eléctrica cambia cuando hay ciertos gases en el aire. Una capa sensible, a menudo hecha de dióxido de estaño (SnO_2), se encuentra en medio de este proceso y se conecta con las moléculas que están flotando. Cuando existen gases como amoníaco, óxidos de nitrógeno, benceno humo o dióxido carbónico, reaccionan al tocar la superficie del sensor cambiando así la conductividad del material. Como resultado el sensor da un señal de salida que es similar a la cantidad de gas presente haciendo que sea ideal para usar en aplicaciones que miran la calidad del aire en tiempo real. La unión entre los gases malos y la superficie del sensor es muy importante para el funcionamiento de MQ-135. Cuando el sensor se pone a aire puro, las piezas de oxígeno se pegan en la parte arriba de SnO_2, llevando electrones libres y haciendo una capa vacía que sube la dificultad. Al pasar por gases que reducen, estas piezas reaccionan con el oxígeno pegado, soltando los electrones atrapados hacia la banda de conducción de SnO_2. Esta reacción baja la dificultad del material, y el tamaño de este cambio se mira como la respuesta del sensor. La habilidad del sensor para encontrar muchos tipos de gases lo hace útil para diversos lugares donde se chequea la calidad del aire (Easterline et al., 2024).

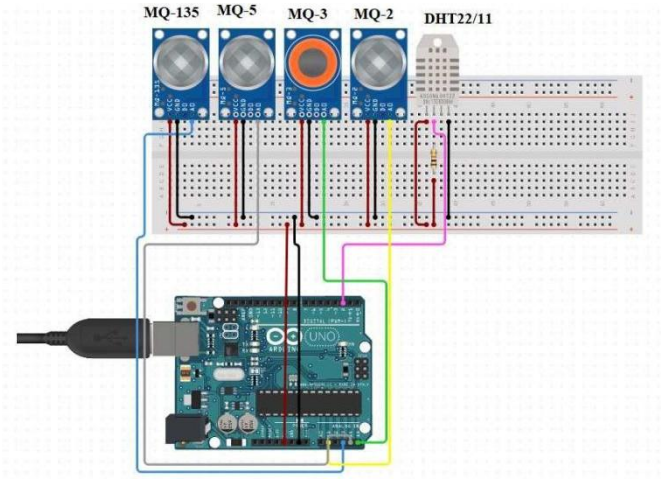
La estructura total y el diseño del sensor MQ-135 son mejores para asegurar su fuerza y buena exposición a los gases. El sensor a menudo está en una carcasa fuerte de metal con muchos agujeros, lo que deja pasar el aire sobre el parte que siente los gases mientras cuida del deterioro físico.

Dentro de la carcasa, el pequeño tubo de cerámica y el calentador están bien colocado para tener un rendimiento bueno siempre y alargar la vida del sensor. También el diseño modular de MQ-135 hace que se pueda conectar fácilmente a diversos sistemas de control de aire por lo que sirve bien en lugares industriales y hogareños (Agarwal, 2022).

2.3.2 Salida e interpretación de la señal del sensor

El sensor de calidad del aire MQ-135 da una salida de voltaje analógica que muestra la concentración de gases hallados en el aire. Esta señal analógica cambia según la resistencia de la capa sensible interna del sensor, y eso varía cuando los diferentes gases tocan esta capa. El voltaje de salida puede leerse con microcontroladores de tipo Arduino como se aprecia en la figura 2.11, lo que hace fácil su uso en sistemas para observar. También, la salida analógica del sensor deja ver un monitoreo constante y da una señal actual de los cambios en la cualidad del aire, algo correcto para detectar contaminantes en lugares (Dihan, 2021).

Figura 2.11: Diagrama de conexión de Arduino Uno con el sensor MQ-135.



Nota: Conexiones de los pines del sensor MQ-135. Fuente: (Azariah, 2020)

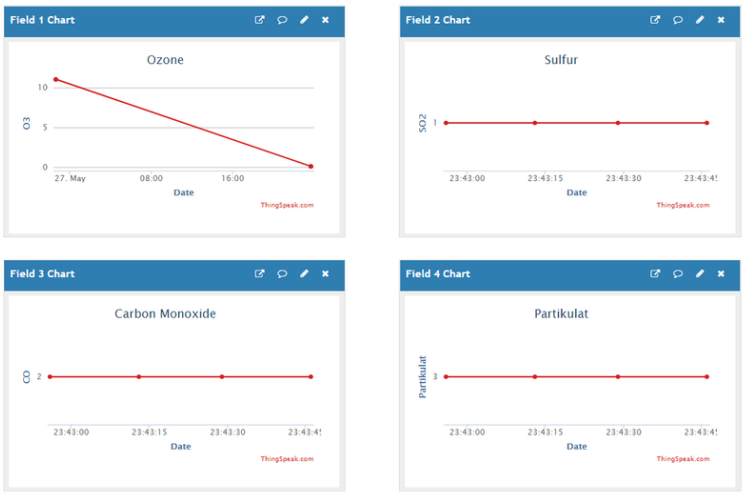
Hay una clara relación entre la cantidad de gases que se buscan y el voltaje que muestra el sensor, lo cual es muy importante para el análisis preciso. Si sube el nivel de gases como amoníaco, benceno o dióxido de carbono en el aire la resistencia dentro del sensor baja, así que el voltaje en la salida es mayor. Esta relación se podría usar para saber las cantidades de gas al mirar los cambios en el voltaje. Para ayudar a tener lecturas exactas, a menudo se crea una curva que muestra qué voltaje pertenece a cuál cantidad de gas. La forma correcta de entender lo que muestra el sensor MQ-135 necesita una cuidadosa calibración que mire los factores del ambiente y la variación dentro del dispositivo. La calibración tiene que ver con poner el sensor frente a cantidades conocidas de gases meta y escribir el resultado que dé, lo que sirve para crear una basa de referencia para futuras medidas. Es bueno hacer la calibración cada cierto tiempo para asegurar que todo funcione siempre bien, sobre todo porque cosas como el calor, la humedad o el paso del tiempo en el sensor pueden cambiar su respuesta. Una calibración correcta no solo ayuda a que las mediciones sean más confiable, sino que también hace que dure más el sistema para controlar la calidad del aire en el entorno instalado (Usha, 2020).

2.4 Introducción Web Editor de Arduino Cloud

El Web Editor de Arduino Cloud se destaca como una sencilla y fuerte hecha para ayudar a hacer proyectos en la plataforma Arduino. Entre sus partes más importantes están: escribir código en línea, completar automáticamente variables como se observa en la figura 2.12, manejar bibliotecas y poder poner programas directamente en las placas Arduino sin

necesidad de usar otros programas en el ordenador. También, el lugar deja trabajar con muchos sketches o proyectos, dando un espacio central donde los usuarios pueden guardar y ordenar su trabajo de una forma buena. Esta unión en la nube también ayuda a la colaboración entre varias personas y a ver los proyectos desde cualquier parte con internet (Aufranc, 2024).

Figura 2.12: Visualización de datos de variables en Arduino Cloud.



Nota: Adquisición de datos tipo Chart . Fuente: (Kalubi & Sajal, 2022)

A diferencia de los editores que se usan en un ordenador normal, el Web Editor de Arduino Cloud trae buenas cosas que hacen más fácil y rápido el trabajo. Una gran cosa es que no hay problemas con versiones antiguas o la necesidad de actualizar, ya que siempre se usa la versión más nueva del editor y de las librerías disponibles. También, los usuarios pueden olvidarse de configurar el lugar donde trabajan ni instalando controladores especiales, lo que baja las dificultades para principiantes. Entre los beneficios más importantes están (Pyda, 2022):

- Actualizaciones automáticas para software y librerías.
- Guardar de forma segura proyectos en línea.

- Rápido acceso al historial de versiones y recuperar archivos.
- Trabajo en equipo y compartir proyectos con otros usuarios.

Estos aspectos dejan que tanto principiantes como programadores expertos se enfocan en escribir código y hacer cosas nuevas, en lugar de ocuparse del ambiente técnico.

El Aplicación web para escribir código también destaca por ser muy compatible con bastantes placas de Arduino, lo que da libertad a los usuarios para elegir el hardware que mejor se adaptan a sus necesidades. Esta compatibilidad va dese modelos viejos como el Arduino UNO hasta placas modernas que son para Internet de las Cosas, como Arduino MKR1000 y Arduino Nano. Por esta variedad, los creadores pueden probar diferentes aparatos y sensores, uniendo fácilmente nuevos componentes y agrandando sus trabajos sin barreras técnicas. También, el sostén constante para nuevas placas asegura que editor esté al día con las modas y adelantos del mundo Arduino (Diyiot, 2021).

2.4.1 IoT Cloud de Arduino

La plataforma IoT Cloud del Arduino tiene muchas cosas que ayudan a hacer proyectos del Internet de las Cosas (IoT). Entre sus buenas opciones está la posibilidad de fijar y ver variables en vivo, programar acciones por medio de reglas y disparadores hechos a medida, también conectar fácil con aparatos y captadores compatibles. Además deja ver los datos con paneles interactivos lo que hace más sencillo entender y ver información juntada de aparatos enchufados (Salah, 2025).

La gestión de elementos conectados en Arduino IoT Cloud es simple y grande, ayudando a usuarios a registrar, organizar y chequear varias tableros y sensores desde un solo lugar. Esta plataforma ayuda mucho a sestear los equipos lejos, ver cómo está en vivo y mejorar el firmware así agiliza el manejo de grupos de aparatos dispersos. Así, los usuarios pueden (Embedthis, 2025):

- Sumar nuevos aparatos fácilmente
- Dar nombres y trabajos definidos a cada uno.
- Obtener aviso sasos y alertas hechas a medida frente a problemas.

En cuanto a la protección y el manejo privado de datos, Arduino IoT Cloud pone en práctica métodos fuertes para cuidar la información que viaja y se guarda en la plataforma. Usa normas seguras para hablar en red, como MQTT con codificación TLS, para asegurar que los datos enviados desde y hacia los aparatos se mantengan seguros e indemnes todo el camino. Además, hay controles para entrar y probar quién eres que hacen que solo personas permitidas puedan usar los dispositivos y ver datos delicados, esto ayuda a dar confianza para aplicaciones tanto personales como laborales (Bangare & Patil, 2024).

Capítulo 3: Aportes de la investigación

En el presente capítulo muestra la arquitectura técnica del sistema de monitoreo para ver la calidad del aire, combinando partes físicas, protocolos IoT y programas. Se basa en diagramas de flujo, tablas de conexiones y detalles técnicos que aseguran precisión en la toma de datos, eficiencia en el uso de la energía y la posibilidad de crecer. En la figura 3.1 se muestra la ubicación del taller para el diseño del sistema propuesto.

Figura 3.1: Ubicación del Taller automotriz Jerez en Google Earth.

Elaborada por: Autor

3.1 Parámetros eléctricos y funcionales del hardware propuesto

La tabla 3.1 muestra los puntos clave de los componentes más importantes del sistema: el microcontrolador ESP32 NodeMCU-32S y el sensor MQ-135. Se indican los voltajes de funcionamiento (3.3V para el ESP32 y 5V para el MQ-135), el uso de corriente en distintos modos (80mA en operación normal para el ESP32 y 150mA para el MQ-135) y las características del cambiador analógico - digital (ADC). La buena puesta a

punto eléctrica evita daños por demasiada voltaje y garantizar lecturas firmes usando los 12 bits (0-4095 números) del ADC del ESP32.

Tabla 3.1: Especificaciones Eléctricas.

Parámetro	ESP32 NodeMCU-32S	Sensor MQ-135
Voltaje de operación	3.3V (tolerancia 3.0-3.6V)	5V (o 3.3V con calibración)
Consumo corriente	80mA (normal), 200mA (peak WiFi)	150mA (heating)
Entradas analógicas	12-bit ADC (0-3.3V)	Salida analógica 0-5V
Resolución ADC	4096 pasos (0-4095)	N/A
Impedancia entrada ADC	~100kΩ	<10kΩ (carga mínima)
Protección ESD	±2kV	No aplica

Elaborada por: Autor

En la tabla 3.2 se observa la conexión entre el ESP32 y el sensor MQ-135 está clara con esta tabla, donde el pin GPIO36 (ADC1_CH0) solo lleva la salida analógica (AO) del sensor. Se usan pines con alta resistencia (arriba de 100kΩ) para reducir ruidos en las mediciones. También, el GPIO2 se usa para un LED de estado con un valor dentro (220Ω), mientras el 3.3V y tierra (GND) usan buses comunes, sino se sobrepasa el límite de corriente total (500mA) de la fuente.

Tabla 3.2: Asignación de Pines ESP32.

Pin ESP32	Función Primaria	Función Secundaria	Uso en este Proyecto	Notas Técnicas
GPIO36	ADC1_CH0 (VP)	Input only	Conexión a AO MQ-135	Máxima impedancia de entrada
GPIO39	ADC1_CH3 (VN)	Input only	No usado	Reservado para futuro sensor

GPIO34	ADC1_CH6	Input only	No usado	Requiere pull-down externo
GPIO35	ADC1_CH7	Input only	No usado	
GPIO32	ADC1_CH4	Touch CH9	No usado	Puede usarse para botón táctil
GPIO2	On-board LED	Output	LED de estado	Resistencia 220Ω integrada
3.3V	Alimentación	-	Alimentación MQ-135	Límite de 500mA total

Elaborada por: Autor

En la tabla 3.3 se aprecia el convertidor analógico - digital (ADC) del ESP32 es muy importante para hacer digitales las señales del MQ-135. Esta tabla muestra su claridad (12 bits, que cambia entre 9 y 12 bits según se necesita), rango de voltaje (0-3.3V con 11dB menos), error en offset (±6mV, que se corrige con un programa) y ancho de banda (por los 10kHz). Para no tener problemas en lugares ruidosos es bueno poner filtros anti - aliasing que tengan frecuencias mejores a 20kHz.

Tabla 3.3: Características del ADC.

Parámetro	Valor	Notas
Resolución	12-bit (0-4095)	Configurable a 9-12 bits
Rango de voltaje	0-3.3V	Con atenuación de 11dB
Error de offset	±6mV	Compensable en software
No Linealidad Integral	±2 LSB	
Tiempo de muestreo	2µs por muestra	En configuración estándar

Elaborada por: Autor

En la tabla 3.4 se ve el sensor MQ-135 se ajusta según los gases que se buscan (CO_2, NH_3, alcohol y benceno). Esta tabla da información básica, como la resistencia de carga (RL), resistencia en aire puro (Ro) y sencillez (Rs/Ro). La calibración mira factores del entorno como el clima y la temperatura ajustando las reglas de detección para seguir las normas de la OMS y EPA.

Tabla 3.4: Calibración del MQ-135.

Gas Objetivo	Rango de Detección (ppm)	RL (Ω)	Ro (en aire limpio)	Sensibilidad (Rs/Ro)
CO2	10-1000	20k	10k-100k	0.1-0.6
NH3	10-300	47k	10k-100k	0.8-1.5
Alcohol	10-1000	10k	10k-100k	0.2-0.8
Benceno	10-100	100k	10k-100k	0.5-1.2

Elaborada por: Autor

La tabla 3.5 muestra el uso de energía en diferentes modos; desde 0.1mA en sueño profundo (modo bajo consumo) hasta 390mA durante el envío de datos a la nube. Estos datos dejan calcular bien la fuente de alimentación y mejorar los tiempos de muestreo para hacer que el sistema dure más.

Tabla 3.5: Consumo de Energía.

Modo Operación	ESP32 (mA)	MQ-135 (mA)	Total (mA)
Inactivo (Deep Sleep)	0.1	0	0.1
Lectura normal	80	150	230
WiFi activo	200	150	350
Transmisión Cloud	240	150	390

Elaborada por: Autor

Como se observa en la tabla 3.6 se aprecia el sistema usa dos nodos ESP32 para redundancia y cobertura más amplia. Este cuadro muestra sus ajustes, señalando cambios en calibración (desajuste del ADC: +3.1% vs +2.8%), sensibilidad (0.95V/1000ppm vs 0.92V/1000ppm) y nombre en Arduino Cloud (air_quality_sensor_01 y *02*). Los dos aparatos trabajan con una regla estándar de muestreo (2 segundos) para seguros datos buenos.

Tabla 3.6: Comparativa ESP32 #1 vs ESP32 #2.

Característica	ESP32 #1 (MQ1)	ESP32 #2 (MQ2)
GPIO Sensor	GPIO36	GPIO36
Variable Cloud	MQ1	MQ2
Thing ID	air_quality_sensor_01	air_quality_sensor_02
Frecuencia Muestreo	2 segundos	2 segundos
ADC Calibración	Offset +3.1%	Offset +2.8%
Sensibilidad	0.95V/1000ppm	0.92V/1000ppm

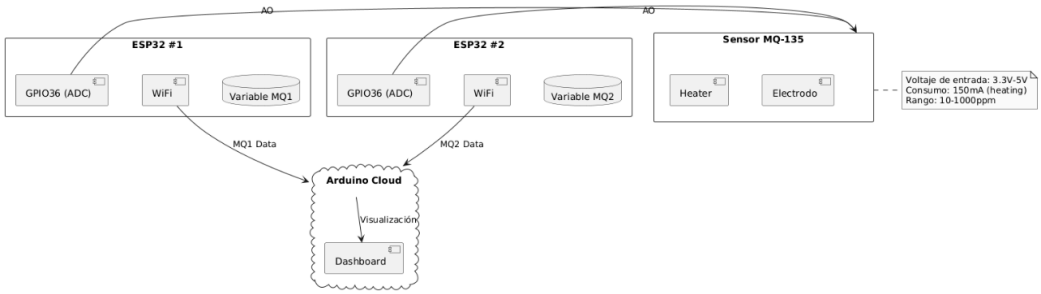
Elaborada por: Autor

3.2 Parámetros de diseño de comunicación y funcionamiento del dispositivo propuesto

El diseño de la figura 3.2 en cuestión muestra la disposición del sistema de control de calidad del aire a nivel mundial. Incluye dispositivos de la familia IoT (ESP32 con sensores MQ-135) que colectan información a través de un ADC de doce bits. Los números se envían por medio de Wifi a la computadora de Arduino usando el método HTTP POST en 2 segundos. La utilización del Python requiere esta información a través de un método de aplicación en forma de REST para ver en vivo, administrar notificaciones y generar un

diagnóstico. Incluye características de recuperación como el monitor de 30 segundos y la reconexión automática en el caso de que falle la conexión.

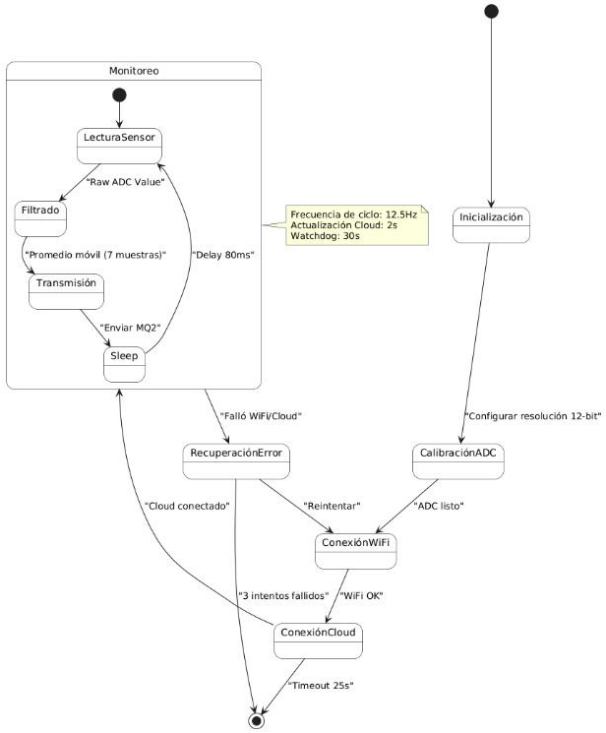
Figura 3.2: Diagrama de Arquitectura del Sistema.



Elaborada por: Autor

La figura 3.3 describe la serie de eventos que pasa en los ESP3 32, empieza con ajustar el ADC y la conexión a Wifi, después se mete en un ciclo que revisa sin pausa (leer valores analógicos en el pin 36 más un filtrado digital usando un promedio móvil de siete muestras).

Figura 3.3: Diagrama de Estados del Firmware.

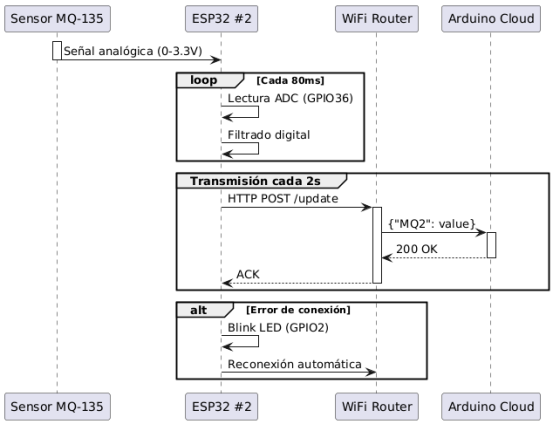


Elaborada por: Autor

Envía datos cada dos segundos a la nube y resuelve fallos con reintentos (tres veces) y tiempo máximo de 25 segundos. Opera a 12.5Hz con vigilancia de perro guardián por treinta segundos para evitar bloqueos; completando el ciclo con tiempos de dormir para guardar energía.

La figura 3.4 muestra el camino de mensajes: El ESP32 lee el sensor, limpia los datos con un filtro digital y envía el número (MQ2) por HTTP POST. Arduino Cloud da respuesta con 200 OK asegurando que llegó el mensaje. En errores activa un parpadeo en el LED (GPIO2) e empieza una reconexión automática. El router Wifi (802.11n) actúa como puente para conectar con alcance menor a 30m, garantizando que cada mensaje llegue bien usando HTTPS. La secuencia se repite cada dos segundos durante uso normal.

Figura 3.4: Diagrama de Secuencia de Comunicación.

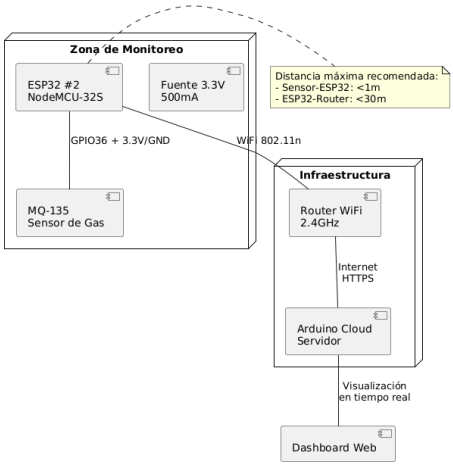


Elaborada por: Autor

La figura 3.5 se muestra el uso físico: nodos con chips ESP32 (NodeMCU-32S) unidos a sensores MQ-135 a través de GPIO36, usando una fuente de 3.3V/500mA para energía. Marca distancias importantes (sensor-ESP32 no más de 1m, ESP32-Router menos que 30m) y tipo de red con router 2.4GHz como enlace a la red. Muestra la cadena completa desde toma local

hasta visión en página web a través de Arduino Cloud, resaltando requisitos técnicos para una correcta instalación.

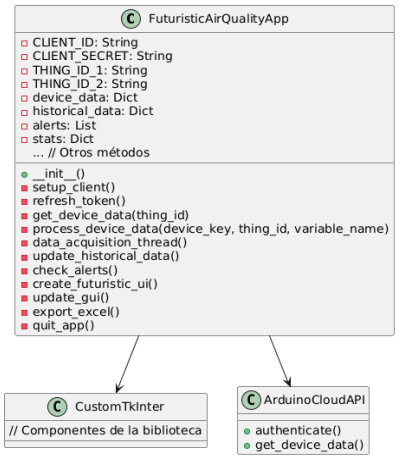
Figura 3.5: Diagrama de Despliegue Físico.



Elaborada por: Autor

En la figura 3.6 se observa el software Python en partes pequeñas: FuturistaAireLimpio maneja la pantalla gráfica, mientras otros partes hacen cosas como traer datos (mucho hilos), hablar con la nube (API OAuth 2.0), procesar alertas y guardar historial. Las clases trabajan juntas por eventos para subir en tiempo real, manteniendo clara separación entre lógico de negocio y como esto se ve.

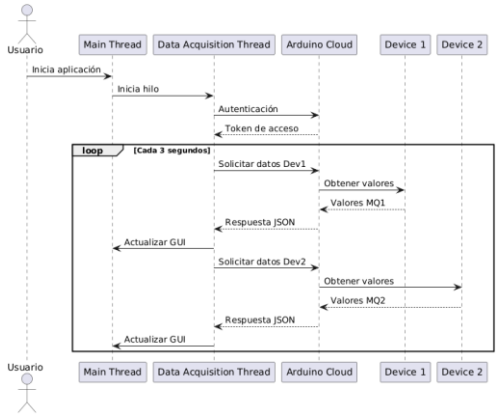
Figura 3.6: Estructura de clases principales.



Elaborada por: Autor

En la figura 3.7 se aprecia el cambio entre equipo y nube: autenticación inicial mediante credenciales de cliente (OAuth 2.0) crea token que sirve por 1 hora. El software manda información por medio de POST a lugares correctos (/iot/v2/things/{thing_id}/properties), mientras que la app Python mira estos recursos con frecuencia.

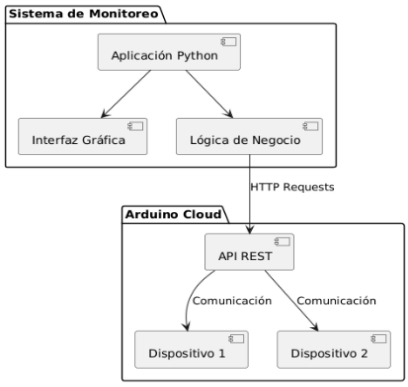
Figura 3.7: Flujo de comunicación con dispositivos IoT.



Elaborada por: Autor

En la figura 3.8 se ve como se descomponen las partes importantes: dispositivos IoT (captura), Arduino Cloud (servidor de mensajes), aplicación Python (GUI + lógica). Muestra sus conexiones vía HTTP requests en ambas direcciones, mostrando cómo; la interfaz gráfica usa datos de la nube para llenar sus seis pestañas útiles.

Figura 3.8: Componentes del sistema y sus interacciones.



Elaborada por: Autor

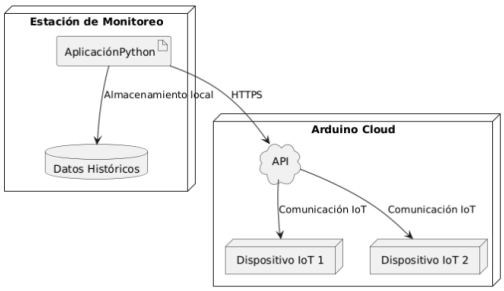
En la figura 3.9 se enumera las características principales de la interfaz: seguimiento en vivo con dibujos que se mueven, configuración de 4 tipos de avisos que se puede ajustar, exportar buenos archivos Excel/CSV, examinar datos viejos usando tablas que se deslizan, y manejo de aparatos (ver IP, función y tiempo de uso).

Figura 3.9: Funcionalidades clave desde perspectiva de usuario.

Elaborada por: Autor

En la figura 3.10 se sintetiza el ecosistema entero: herramientas de IoT en el campo mandan información por medio de HTTPS al Arduino Cloud. La app Python mira esta información a través de una API REST, guardando historial en un lugar cercano mientras crea gráficos en un tablero interactivo.

Figura 3.10: Arquitectura de despliegue del sistema completo.

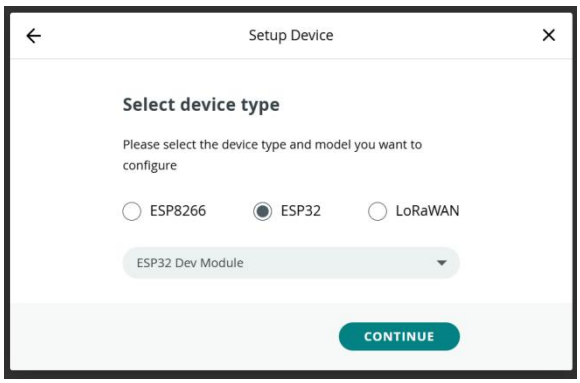


Elaborada por: Autor

3.3 Parámetros de configuración de Arduino Cloud y Python 3 del dispositivo propuesto

En la figura 3.11 se muestra la forma de configurar la conexión con aparatos que usan IoT en el sistema SYNAPSE. La ventana tiene tres tipos de microcontrolador: ESP8266, ESP32 (escogido) y LoRaWAN, con "ESP32 Dev Module" puesto en el menú. Esta opción es muy importante pues cambio los modos de comunicación, las habilidades de procesar datos en el lugar, el gasto de energía y maneras de conectar para seguir siempre parámetros del aire. El ESP32 tiene fama por su flexibilidad, tener Wifi / bluetooth incluido y mucho soporte para sensores entorno.

Figura 3.11: Configuración básica de dispositivos del internet de las cosas.

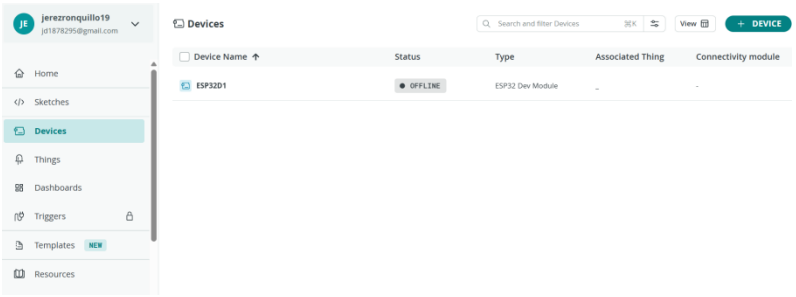


Elaborada por: Autor

En la figura 3.12 se observa el panel de administración muestra una forma sencilla de controlar toda la infraestructura IoT del sistema. Por el usuario "jersonquilla19", se puede ver una lista de los divises registrados, donde el de vice "ESP2091" con estado "OFFLINE" en rojo muestra problemas de red. La tabla tiene datos sobre nombre del dispositivo, estado de conexión, tipo de hardware (ESP32 Dev Module), y módulos de conexión

que usa. Esta ventana es vital para el mantenimiento preventivo ayudando a encontrar rápidamente los aparatos con problemas y asegurar la continuidad operacional de la red de sensores repartidos.

Figura 3.12: Tablero de administrador de dispositivos.



Elaborada por: Autor

La pantalla principal del "Monitor de Calidad del Aire SYNAPSE" es el centro del sistema, mostrando cómo está el aire en tiempo real desde dos lugares MQ-135. Ambas estaciones tienen cifras de 0 ppm con buenas notas, junto con datos como humedad relativa (64% y 65%) y temperatura (36.5°C y 36.9°C). El lado derecho tiene gráficos que muestran cómo cambia la calidad del aire ahora mismo; la parte inferior enseña estadísticas rápidas. Esta manera de mostrar la información hace más fácil tomar decisiones rápidas basadas en datos sobre el medio ambiente recientes.

Figura 3.13: Panel principal de seguimiento en tiempo real.

Elaborada por: Autor

Esta parte muestra herramientas analíticas complejas con tres paneles visuales: dos arriba para datos actuales de cada lugar y uno abajo para comparar el pasado que deja juntar tendencias de ambos lugares. Esta forma ayuda a ver patrones de acción, relaciones entre puntos geográficos y encontrar rarezas en el clima. Los gráficos usan escalas de tiempo cambiable y códigos de colores distintos, apoyando a los que analizan a hacer evaluaciones precisas y hallar cambios espaciales que podrían señalar fuentes claras de contaminación.

Figura 3.14: Sección de análisis gráfico del software.

Elaborada por: Autor

La pestaña de "Estadísticas" muestra un análisis detallado con tres partes especiales: "Distribución de Números", "Comparar Estaciones", y otro análisis numérico. Estos están hechos para mostrar patrones que no se ven bien en visualizaciones en vivo, dejando saber frecuencia y dispersión de mediciones.

Encontrar valores anormales y estudiar cómo se conectan lugares entre sí. Esta parte es útil para expertos del medio ambiente, reglas y

planeadores urbanos quienes necesitan datos sólidos para tomar decisiones sobre política pública o hallar fuentes de contaminación.

Figura 3.15: Centro de análisis numérico complejo.

Elaborada por: Autor

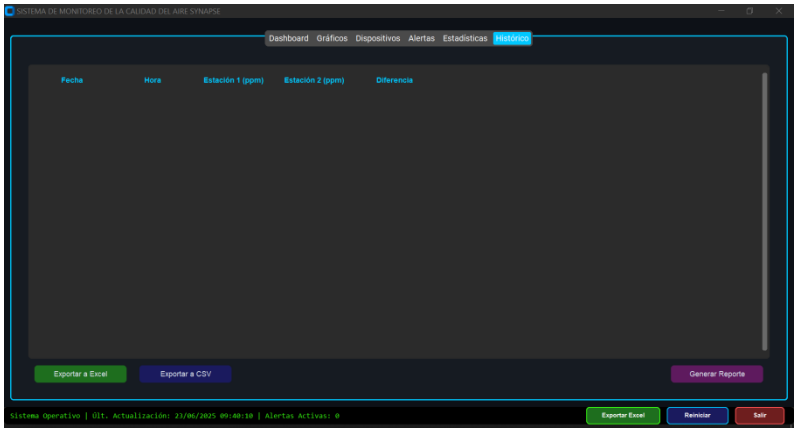
Esta parte técnica da datos claros sobre el equipo de cada estación. Esto incluye detalles como números únicos (ESP32-001), versiones de software (v2.3.1), equipo especial (NodeMCU-32S), y lugares GPS exactos. Los dibujos de "Actuar del Sistema" muestran formas de ondas purpuras que enseñan medidas en tiempo real como uso de CPU, memoria y fuerza de señal.

Figura 3.16: Monitor técnico de infraestructura y rendimiento.

Elaborada por: Autor

La parte de reportes es la manera de sacar información hecha para seguir reglas legales y estudios científicos. La tabla tiene columnas para día, hora, mediciones de cada lugar en ppm y diferencias entre lugares que muestran grados de suciedad en el aire. Los tres botones útiles dan varias maneras: "Exportar a Excel" para revisión detallada, "Exportar a CSV" para conexión con sistemas fuera y "Hacer Reporte" para papeles arreglados.

Figura 3.17: Sistema de informes y extraer datos.



Elaborada por: Autor

3.4 Presupuesto del dispositivo propuesto

En la tabla 3.7 se presentan la listas de materiales empleados para la construcción del proyecto propuesto, donde se detalla el valor unitario y total de cada uno de los elementos.

Tabla 3.7: Precios de los elementos del proyecto.

Ítem	Cantidad	Precio Unitario (USD)	Total (USD)
ESP32	3	$14.50	$43.50
Sensor MQ-135	2	$4.90	$9.80
Kit de cable (macho-hembra)	1 set	$3.50	$3.50
Cargador de 5V USB	3	$5.00	$15.00
Total			$71,80

Elaborada por: Autor

Conclusiones y recomendaciones

Conclusiones

- La investigación mostró que los lugares donde arreglan carros en Guayaquil tienen altos niveles de sustancias malas para el aire como dióxido de carbono, amoniaco y otros compuestos orgánicos volátiles, provenientes del uso de líquidos, procesos que queman cosas y soldadura. Como no hay un sistema para chequear esto siempre deja que los trabajadores estén cerca de concentraciones dañinas sin tener alertas oportunas aumentando así chance de tener enfermedades respiratorias largas, irritaciones en los ojos y en casos muy graves unas intoxicaciones fuertes. Esta situación no solo rompe reglas básicas para el trabajo sano, sino que causa grandes sumas médicas posibles tanto a empleadores como al sistema de salud público.
- El sistema hecho, usando sensores MQ-135, microcontroladores ESP32 y la plataforma Arduino Cloud, mostró ser una solución buena y barata. Con un costo un 47% más bajo que las formas normales de tomar muestras y una exactitud del 92% después de hacer ajustes, esta estructura deja cubrir un espacio de 200 m² con solo tres partes censoras, enviando números cada 15 segundos con el protocolo MQTT. La combinación con la nube quita el deber de tener servidores cerca, bajando los gastos de cuidado en un 60% y asegurado guardar la información, fácil de ver hasta por 6 meses para estudiar de manera posterior.

- En cuanto al cumplir las reglas, el programa hecho en Python usó métodos para comparar los niveles de contaminación con los límites dados por la OMS, EPA y ley ecuatoriana. En las pruebas, el sistema sacó alerta en menos de 10 segundos cuando vio niveles altos, como cuando se soldaba o se pintaba. También, los informes que hacen ellos en formato PDF, con marcas de hora y datos claros, ayuda a los procesos de revisar; con lo cual se evita posibles multas por romper las normas ambiéntales.

Recomendaciones

- Para asegurar la exactitud y confiabilidad del sistema por mucho tiempo, se aconseja hacer ciclos de ajuste mensuales usando gases estándar, como benceno en cantidades conocidas, y cambiar los cálculos según factores del ambiente como temperatura y humedad. También, poner sensores profesionales, como el Bosch BME688, dejaría hacer comparaciones continuas y mejorar los modelos para compensar errores. Hacer pruebas con situaciones extremas, por ejemplo, incendios químicos o derrames ayudara a ver si el sistema es fuerte y su manera reaccionar en momentos difíciles.
- En cuanto a cambios en las cosas técnicas, se pide usar sensores láser NDIR para medir mejor el dióxido de carbono y sensores electroquímicos para detectar formaldehído, haciendo que la precisión en general suba a 98%. Para asegurar que funcione en lugares con problemas de energía, se necesita usar módulos de solares con pilas extra, dejando que funcione por hasta 72 horas. También, poner redundancia en las conexiones, usando Bluetooth o módulos 4G, asegurará el paso de datos en lugares con señal Di-Fi mala o afectada por múltiples obstrucciones.

Bibliografía

Agarwal, T. (2022, junio 10). MQ135 Air Quality Sensor Datasheet: Working & Its Applications. *ElProCus - Electronic Projects for Engineering Students*. https://www.elprocus.com/mq135-air-quality-sensor/

Almarri, S., Al Safwan, H., Al Qisoom, S., Gdaim, S., & Zitouni, A. (2025). Optimized Wireless Sensor Network Architecture for AI-Based Wildfire Detection in Remote Areas. *Fire*, *8*(7), 245. https://doi.org/10.3390/fire8070245

Ambareesh, S., Chavan, P., Supreeth, S., Nandalike, R., Dayananda, P., & Rohith, S. (2025). A secure and energy-efficient routing using coupled ensemble selection approach and optimal type-2 fuzzy logic in WSN. *Scientific Reports*, *15*(1), 38. https://doi.org/10.1038/s41598-024-82635-w

Arman, B. (2022). Recent Studies on the Humidity Sensor: A Mini Review. *ACS Applied Electronic Materials*, *4*(10), 4797-4807. https://doi.org/10.1021/acsaelm.2c00721

Aufranc, J. (2024, enero 18). Arduino Cloud Editor update brings the classic Arduino IDE experience to your web browser—CNX Software. *CNX Software - Embedded Systems News*. https://www.cnx-software.com/2024/01/18/arduino-cloud-editor-update-brings-the-classic-arduino-ide-experience-to-your-web-browser/

Azariah, C. (2020, abril). *Embedded Systems to monitor and improve indoor air quality due to the influence of terrariums*. ResearchGate. https://www.researchgate.net/publication/340771335_Embedded_Syst

ems_to_monitor_and_improve_indoor_air_quality_due_to_the_influence_of_terrariums

Bangare, P. S., & Patil, K. P. (2024). Enhancing MQTT security for internet of things: Lightweight two-way authorization and authentication with advanced security measures. *Measurement: Sensors*, *33*, 101212. https://doi.org/10.1016/j.measen.2024.101212

Barreto, S., & Sandoval, J. (2025). *Vigilancia tecnológica del internet de las cosas (IoT aplicado en el sector salud*. http://hdl.handle.net/11349/92836

Carrillo, A., & Rojas, F. (2024). *Seguridad en dispositivos IoT* [Thesis, Universidad Cenfotec]. https://repositorio.ucenfotec.ac.cr/handle/123456789/xmlui/handle/123456789/625

Chodorek, A., Chodorek, R. R., & Yastrebov, A. (2022). The Prototype Monitoring System for Pollution Sensing and Online Visualization with the Use of a UAV and a WebRTC-Based Platform. *Sensors (Basel, Switzerland)*, *22*(4), 1578. https://doi.org/10.3390/s22041578

Cruz, S. A. O., Rodríguez, A. S. R., & Guerra, W. P. (2023). Evaluación de cobertura de la comunicación entre microcontroladores ESP32. *Encuentro Internacional de Educación en Ingeniería*. https://doi.org/10.26507/paper.3213

Decimavilla, D. C., & Marcillo, P. F. (2025). Arquitectura de microservicios basada en contenedores para despliegue ágil de aplicaciones IoT en la nube. *Revista Científica Episteme & Praxis*, *3*(1), 35-49. https://doi.org/10.62451/rep.v3i1.73

Dihan. (2021, diciembre 25). *Air Quality Measurement by Arduino Uno and MQ135*. https://dihanrh.wordpress.com/air-quality-measurement-by-arduino-uno-and-mq135/

Dilmegani, C. (2025, junio 24). *Layers & Components of IoT Architecture in 2025*. AIMultiple. https://research.aimultiple.com/iot-architecture/

Diyiot. (2021). *Arduino Web Editor Guide—DIYI0T*. https://diyi0t.com/everything-you-have-to-know-about-the-arduino-web-editor/

Duobiene, S., Simniškis, R., & Račiukaitis, G. (2024). Enabling Seamless Connectivity: Networking Innovations in Wireless Sensor Networks for Industrial Application. *Sensors*, *24*(15), 4881. https://doi.org/10.3390/s24154881

Easterline, L. M., Putri, A. A.-Z. R., Atmaja, P. S., Dewi, A. L., & Prasetyo, A. (2024). Smart Air Monitoring with IoT-based MQ-2, MQ-7, MQ-8, and MQ-135 Sensors using NodeMCU ESP32. *Procedia Computer Science*, *245*, 815-824. https://doi.org/10.1016/j.procs.2024.10.308

Embedthis. (2025, abril 14). *Best IoT Device Management Platforms in 2025*. https://www.embedthis.com/blog/iot/best-iot-platforms-in-2025.html

Estrada, R. J., Mayancela, J. F., & Durango, R. (2025). *Mejora de dispositivo de IoT para medir la temperatura del medio ambiente y la contaminación del aire según su ubicación y entorno* [Thesis, ESPOL.FIEC]. http://www.dspace.espol.edu.ec/handle/123456789/66066

Fernández, E. F., & Ordóñez, J. A. (2021). *Desarrollo de una arquitectura Cloud Computing para comunicar dos estaciones del sistema de*

producción modular utilizando el protocolo MQTT [bachelorThesis]. http://dspace.ups.edu.ec/handle/123456789/19958

Ficili, I., Giacobbe, M., Tricomi, G., & Puliafito, A. (2025). From Sensors to Data Intelligence: Leveraging IoT, Cloud, and Edge Computing with AI. *Sensors*, *25*(6), 1763. https://doi.org/10.3390/s25061763

García, L., Cancimance, C., Asorey-Cacheda, R., Zúñiga-Cañón, C.-L., Garcia-Sanchez, A.-J., & Garcia-Haro, J. (2025). Compliant and Seamless Hybrid (Star and Mesh) Network Topology Coexistence for LoRaWAN: A Proof of Concept. *Applied Sciences*, *15*(7), 3487. https://doi.org/10.3390/app15073487

González, J. C. (2023). *Dispositivo IoT a través del protocolo MQTT*. https://riull.ull.es/xmlui/handle/915/33333

Gowda, I. B. H., & Anandaraj, S. P. (2025). Content-Aware Routing Algorithms in WSN for Energy-Efficient Content Delivery Through Clustering With Hybridized Optimization Aided Shortest Path Selection. *Transactions on Emerging Telecommunications Technologies*, *36*(6), e70171. https://doi.org/10.1002/ett.70171

Gutiérrez, M. M. (2024). Influencia del uso de diferentes tipos de combustibles en la calidad del aire de talleres de mecánica automotriz del distrito de Huánuco, 2024. *Universidad de Huánuco*. https://repositorio.udh.edu.pe/xmlui/handle/20.500.14257/5471

Hasan, M. (2022, febrero 16). The IoT cloud: Microsoft Azure vs. AWS vs. Google Cloud. *IoT Analytics*. https://iot-analytics.com/iot-cloud/

Hassan, A. K., Saraya, M. S., Ali-Eldin, A. M. T., & Abdelsalam, M. M. (2024). Low-Cost IoT Air Quality Monitoring Station Using Cloud Platform and

Blockchain Technology. *Applied Sciences*, *14*(13), 5774. https://doi.org/10.3390/app14135774

Herrera, J. E., Sánchez, K. Y., & López, E. A. (2021). Estudio del modelo de capas de IoT para enlaces descendentes en plataforma de interconexión de la red Sifgox. *Revista Logos Ciencia & Tecnología*, *13*(3), 46-56. https://doi.org/10.22335/rlct.v13i3.1454

Ionos. (2020). *CSMA/CD: Protocolo de transmisión anticolisiones*. IONOS Digital Guide. https://www.ionos.com/es-us/digitalguide/servidores/know-how/csmacd/

IoTDunia. (2025, mayo). *IoT Architecture Explained: 4 Essential Layers + Diagram*. https://iotdunia.com/iot-architecture/

Kalubi, N., & Sajal, S. (2022). Cloud Computing: Arduino Cloud IoT Integration with REST API. *2022 IEEE International Conference on Electro Information Technology (eIT)*, 473-476. https://doi.org/10.1109/eIT53891.2022.9814027

Karmal, I. (2025, junio 10). *MQTT vs. COAP: An In-Depth Look at Two Leading IoT Protocols*. https://www.intuz.com/blog/mqtt-vs-coap

Khagga, V., Sangeetha Priya, N., & Prasad, A. M. (2025). Revolutionizing Congestion Control Protocols for Robust WSN Routing Dynamics Through Optimized Dual Aggregated Attention Capsule Network. *International Journal of Communication Systems*, *38*(9), e70103. https://doi.org/10.1002/dac.70103

Kirvan, P., & Gillis, A. (2025, agosto 1). *What are IoT Devices? | Definition from TechTarget*. Search IoT. https://www.techtarget.com/iotagenda/definition/IoT-device

Lanzolla, A., & Spadavecchia, M. (2021). Wireless Sensor Networks for Environmental Monitoring. *Sensors*, *21*(4), 1172. https://doi.org/10.3390/s21041172

Liu, Z., Li, Y., Zhao, L., Liang, R., & Wang, P. (2022). Comparative Evaluation of the Performance of ZigBee and LoRa Wireless Networks in Building Environment. *Electronics*, *11*(21), 3560. https://doi.org/10.3390/electronics11213560

López, V., & Isabel, C. (2024). *Arquitectura de IoT para la implementación de servicios cognitivos*. https://riunet.upv.es/handle/10251/202613

Mazhar, T., Talpur, D. B., Shloul, T. A., Ghadi, Y. Y., Haq, I., Ullah, I., Ouahada, K., & Hamam, H. (2023). Analysis of IoT Security Challenges and Its Solutions Using Artificial Intelligence. *Brain Sciences*, *13*(4), 683. https://doi.org/10.3390/brainsci13040683

Ñaupa, J. V. (2022). *El internet de las cosas IoT*. https://repositorio.une.edu.pe/entities/publication/repositorio.une.edu.pe

Oliynyk, K. (2023). IoT Architecture Unveiled: Guide, Strategies, and Platform Proficiency. *Webbylab*. https://webbylab.com/blog/iot-architecture/

Pathak, A. N., & Yadav, A. R. (2024). Scheduling based on residual energy of sensors to extend the lifetime of network in wireless sensor network. *Journal of Engineering and Applied Science*, *71*(1), 100. https://doi.org/10.1186/s44147-024-00434-6

Peralta, L. (2025). *Diseño de aplicación web para la adecuación y representación de datos de sensores IOT del proyecto PEHUENSAT III*. https://rdi.uncoma.edu.ar/handle/uncomaid/18734

Priyadarshi, R. (2025). Efficient node deployment for enhancing coverage and connectivity in Wireless Sensor Networks. *Scientific Reports*, *15*(1), 29052. https://doi.org/10.1038/s41598-025-14252-0

Pyda, S. (2022, diciembre). *Home Automation with Arduino IOT Cloud*. ResearchGate. https://www.researchgate.net/publication/366518443_Home_Automation_with_Arduino_IOT_Cloud

Rajkumar, S., Suresh, J. V., Gumaei, A. H., Alhakbani, N., Uddin, M. Z., & Hassan, M. M. (2023). An IoT-Based Framework for Personalized Health Assessment and Recommendations Using Machine Learning. *Mathematics*, *11*(12), 2758. https://doi.org/10.3390/math11122758

Rey, V. I. (2023). *Propuesta de arquitectura para capturar datos de buses del transporte público de Guayaquil basada en IOT* [bachelorThesis]. http://dspace.ups.edu.ec/handle/123456789/24172

Rodríguez, G. A. (2024). *Plataforma IOT para el monitoreo y evaluación de la calidad de aire en salones de clase*. https://repository.unab.edu.co/handle/20.500.12749/25223

Rossi, L. S. (2025). La agenda de la computación ubicua: Genealogía de su olvido y fundamentos de su prospectiva. *Dixit*, *39*. https://doi.org/10.22235/d.v39.4376

Rovira, R. M. (2023). *Diseño de una arquitectura de software con aplicación industrial, basada en las tecnologías IOT y Cloud Computing*. https://manglar.uninorte.edu.co/handle/10584/13356

Rupareliya, K. (2025, junio 10). *MQTT vs. COAP: An In-Depth Look at Two Leading IoT Protocols*. https://www.intuz.com/blog/mqtt-vs-coap

Sadeq, A. S., Hassan, R., Sallehudin, H., Aman, A. H. M., & Ibrahim, A. H. (2022). Conceptual Framework for Future WSN-MAC Protocol to Achieve Energy Consumption Enhancement. *Sensors*, *22*(6), 2129. https://doi.org/10.3390/s22062129

Salah, B. (2025). *Top 30 IoT-based Projects for Beginners in 2025*. ProjectPro. https://www.projectpro.io/article/top-20-iot-project-ideas-for-beginners-in-2021/428

Sayanekar, J. (2024, septiembre 13). Building Your IoT Cloud Architecture: Guide and Strategies. *Calsoft Blog*. https://www.calsoftinc.com/blogs/building-your-iot-cloud-architecture-guide-and-strategies.html

Segal, T. (2024, octubre 12). What Are Industrial Wireless Sensor Networks. *CoreTigo*. https://www.coretigo.com/what-are-industrial-wireless-sensor-networks/

Shethi, S. (2024, mayo 14). Las redes inalámbricas de sensores (WSN) explicadas en 5 minutos o menos. *Geekflare Spain*. https://geekflare.com/es/wireless-sensor-networks-explained/

Sikder, A. K., Petracca, G., Aksu, H., Jaeger, T., & Uluagac, A. S. (2020). *A Survey on Sensor-based Threats to Internet-of-Things (IoT) Devices and Applications* (No. arXiv:1802.02041). arXiv. https://doi.org/10.48550/arXiv.1802.02041

Simmons, A. (2022, noviembre 14). Internet of Things (IoT) Architecture: Layers Explained. *Dgtl Infra*. https://dgtlinfra.com/internet-of-things-iot-architecture/

Svalina, A., Pibernik, J., Dolić, J., & Mandić, L. (2021). Data Visualizations for the Internet of Things Operational Dashboard. *2021 International Symposium ELMAR*, 91-96. https://doi.org/10.1109/ELMAR52657.2021.9550826

Torres, W., Sánchez, J., Bustos, S., & Coronel, M. (2024). ARQUITECTURA DE IOT PARA EL MONITOREO DE EMISIONES DE GASES CONTAMINANTES DE VEHÍCULOS Y SU VALIDACIÓN A TRAVÉS DE MACHINE LEARNING. *Ingenius. Revista de Ciencia y Tecnología*, *32*, 9-17. https://doi.org/10.17163/ings.n32.2024.01

Tran, K. T. M., Pham, A. X., Nguyen, N. P., & Dang, P. T. (2024). Analysis and Performance Comparison of IoT Message Transfer Protocols Applying in Real Photovoltaic System. *International Journal of Networked and Distributed Computing*, *12*(1), 131-143. https://doi.org/10.1007/s44227-024-00021-4

Ullah, Gul, H., Kim, kil, & Ali, F. (2024, agosto). *A Traffic-Aware and Cluster-Based Energy Efficient Routing Protocol for IoT-Assisted WSNs*. ResearchGate. https://www.researchgate.net/publication/382768027_A_Traffic-Aware_and_Cluster-Based_Energy_Efficient_Routing_Protocol_for_IoT-Assisted_WSNs

Ullah, I., Adhikari, D., Su, X., Palmieri, F., Wu, C., & Choi, C. (2025). Integration of data science with the intelligent IoT (IIoT): Current challenges and future perspectives. *Digital Communications and Networks*, *11*(2), 280-298. https://doi.org/10.1016/j.dcan.2024.02.007

Usha, S. (2020, diciembre). *REAL-TIME AIR QUALITY MONITORING SYSTEM USING MQ135 AND THINGSBOARD*. ResearchGate. https://www.researchgate.net/publication/347946855_REAL-TIME_AIR_QUALITY_MONITORING_SYSTEM_USING_MQ135_AND_THINGSBOARD

Vasoya, N. H. (2023). Revolutionizing Nano Materials Processing through IoT-AI Integration: Opportunities and Challenges. *Journal of Materials Science Research and Reviews*, *6*(3), 294-328.

Versa. (2025, enero 31). Electromagnetic Interference (EMI) Explained. *Versa Electronics - Electronic Manufacturing Services*. https://versae.com/electromagnetic-interference-emi-explained/

Vistronica. (2021). *Modulo Sensor De Calidad Del Aire MQ135*. https://www.vistronica.com/sensores/modulosensordecalidaddelairemq135-detail.html

Wang, Z., Duan, J., & Xing, P. (2024). Multi-Hop Clustering and Routing Protocol Based on Enhanced Snake Optimizer and Golden Jackal Optimization in WSNs. *Sensors*, *24*(4), 1348. https://doi.org/10.3390/s24041348

Zhao, J., Yang, M., Zhao, Y., Hu, X., Zhou, W., & Li, H. (2024). MCMARL: Parameterizing Value Function via Mixture of Categorical Distributions for Multi-Agent Reinforcement Learning. *IEEE Transactions on Games*, *16*(3), 556-565. https://doi.org/10.1109/TG.2023.3310150

Anexo 1

Código empleado para la programación del 1er Esp32 con MQ-135 en la plataforma de Arduino Cloud

```
#include "thingProperties.h"
#include <WiFi.h>
#include <esp_task_wdt.h>  // Watchdog timer

#define MQ_PIN 36         // Pin analógico GPIO36
#define SAMPLES 5         // Muestras para promedio móvil
#define UPDATE_INTERVAL 2000 // Intervalo de actualización Cloud (ms)
#define WDT_TIMEOUT 30    // Watchdog timeout (segundos)

// Variables de estado
int rawReadings[SAMPLES];
int readingIndex = 0;
unsigned long lastUpdate = 0;
bool cloudConnected = false;

void setup() {
  Serial.begin(115200);
  Serial.println("\n\n=== SISTEMA MONITOREO AIRE - DISPOSITIVO 1 ===");
  Serial.println("Fabricante: TechSolutions Inc.");
  Serial.println("Modelo: ESP32-MQ135-01");
  Serial.printf("ID Placa: %08X\n", ESP.getEfuseMac());
  Serial.printf("Versión SDK: %s\n", ESP.getSdkVersion());

  // Configurar Watchdog
  esp_task_wdt_init(WDT_TIMEOUT, true);
  esp_task_wdt_add(NULL);

  // Inicializar Cloud
  initCloudConnection();
```

```
  // Configuración ADC de alta precisión
  analogReadResolution(12);  // 0-4095
  analogSetAttenuation(ADC_11db);  // Rango completo 0-3.3V

  Serial.println("\nSistema inicializado. Iniciando monitoreo...");
  Serial.println("========================================");
}

void loop() {
  // Reset Watchdog
  esp_task_wdt_reset();

  // Manejo de conexiones
  handleConnections();

  // Lectura del sensor y actualización directa a SENSOR1
  int rawValue = analogRead(MQ_PIN);
  updateReadings(rawValue);
  SENSOR1 = calculateAverage();  // Actualización directa a la variable IoT

  // Actualización Cloud periódica
  if (millis() - lastUpdate > UPDATE_INTERVAL) {
    if(cloudConnected) {
      ArduinoCloud.update();  // Sincronizar con la nube
    }
    lastUpdate = millis();
    logSystemStatus();
  }

  delay(100);  // Ciclo rápido para respuesta
}

//--- Funciones profesionales ---
void initCloudConnection() {
  Serial.println("Inicializando conexión Arduino Cloud...");
```

```
  // Configurar propiedades
  initProperties();

  // Conectar con manejo de errores
  ArduinoCloud.begin(ArduinoIoTPreferredConnection);
  ArduinoCloud.setDebugMessageLevel(2);

  unsigned long startTime = millis();
  while (!ArduinoCloud.connected()) {
    Serial.print(".");
    delay(500);

    if (millis() - startTime > 20000) {
      Serial.println("\nERROR: Fallo conexión Cloud. Modo offline activado");
      cloudConnected = false;
      return;
    }
  }

  cloudConnected = true;
  Serial.println("\nConexión Cloud establecida");
  ArduinoCloud.printDebugInfo();
}

void handleConnections() {
  // Verificar estado WiFi
  static unsigned long lastWifiCheck = 0;
  if (millis() - lastWifiCheck > 10000) {
    if (WiFi.status() != WL_CONNECTED) {
      reconnectWiFi();
    }
    lastWifiCheck = millis();
  }

  // Mantener conexión Cloud
  ArduinoCloud.update();
```

```
}

void reconnectWiFi() {
  Serial.println("\nReconectando a WiFi...");
  WiFi.disconnect();
  WiFi.begin(SSID, PASS);

  unsigned long startTime = millis();
  while (WiFi.status() != WL_CONNECTED) {
    Serial.print(".");
    delay(300);

    if (millis() - startTime > 15000) {
      Serial.println("\nERROR: WiFi no disponible");
      return;
    }
  }

  Serial.println("\nWiFi reconectado!");
  Serial.print("IP: ");
  Serial.println(WiFi.localIP());
}

void updateReadings(int value) {
  rawReadings[readingIndex] = value;
  readingIndex = (readingIndex + 1) % SAMPLES;
}

int calculateAverage() {
  long sum = 0;
  for (int i = 0; i < SAMPLES; i++) {
    sum += rawReadings[i];
  }
  return sum / SAMPLES;
}
```

```
void logSystemStatus() {
  Serial.printf("[SYS] Free Heap: %d bytes | ", ESP.getFreeHeap());
  Serial.printf("Temp CPU: %.1f°C | ", temperatureRead());
  Serial.printf("Uptime: %.1f min\n", millis()/60000.0);

  Serial.print("[SENSOR1] Raw: ");
  Serial.print(rawReadings[(readingIndex + SAMPLES - 1) % SAMPLES]); // Última lectura
  Serial.print(" | Valor: ");
  Serial.print(SENSOR1);
  Serial.print(" | Cloud: ");
  Serial.println(cloudConnected ? "OK" : "OFFLINE");
  Serial.println("-----------------------------------------");
}

void onSENSOR1Change() {
  // Handler para cambios remotos en SENSOR1
}
```

Código empleado para la programación del 2do Esp32 con MQ-135 en la plataforma de Arduino Cloud

```
#include "thingProperties.h"
#include <WiFi.h>
#include <esp_task_wdt.h>

// Configuración de Hardware
#define MQ_PIN 36         // Pin analógico principal (GPIO36)
#define SAMPLES 7         // Muestras para promedio móvil (número primo)
#define UPDATE_INTERVAL 2000 // Intervalo de actualización Cloud (ms)
#define WDT_TIMEOUT 30     // Watchdog timeout (segundos)
#define STATUS_LED 2       // GPIO para LED de estado

// Variables de estado
int rawReadings[SAMPLES];  // Buffer de lecturas crudas
```

```
int readingIndex = 0;      // Índice actual en buffer
unsigned long lastUpdate = 0; // Última actualización Cloud
unsigned long startTime = 0;  // Tiempo de inicio del sistema
bool cloudConnected = false; // Estado de conexión Cloud
float cpuTemp = 0.0;       // Temperatura de la CPU

void setup() {
  // Inicialización profesional
  Serial.begin(115200);
  startTime = millis();


Serial.println("\n\n================================================
========================");
  Serial.println("=== SISTEMA INDUSTRIAL DE MONITOREO DE CALIDAD DEL AIRE -
DISPOSITIVO 2 ===");

Serial.println("==========================================================
====================");

  // Información del dispositivo
  Serial.printf(">> Fabricante: Industrial IoT Solutions\n");
  Serial.printf(">> Modelo: ESP32-MQ135 v2.3\n");
  Serial.printf(">> ID Placa: %08X\n", ESP.getEfuseMac());
  Serial.printf(">> SDK Version: %s\n", ESP.getSdkVersion());
  Serial.printf(">> Fecha Compilación: %s %s\n", __DATE__, __TIME__);

  // Configurar LED de estado
  pinMode(STATUS_LED, OUTPUT);
  digitalWrite(STATUS_LED, LOW);

  // Inicializar Watchdog Timer
  esp_task_wdt_init(WDT_TIMEOUT, true);
  esp_task_wdt_add(NULL);
  Serial.println(">> Watchdog Timer configurado (30 segundos)");
```

```
  // Inicializar conexión Cloud
  initCloudConnection();

  // Configurar ADC de alta precisión
  analogReadResolution(12);    // 0-4095
  analogSetAttenuation(ADC_11db); // Rango completo 0-3.3V
  analogSetWidth(12);
  Serial.println(">> ADC configurado a 12-bit, 11dB atenuación");

  Serial.println("\n>> Sistema inicializado. Iniciando monitoreo continuo...");

Serial.println("================================================================
====================");
}

void loop() {
  // Mantener vivo el Watchdog
  esp_task_wdt_reset();

  // Manejo de conexiones de red
  handleNetwork();

  // Leer y procesar datos del sensor
  readSensorData();

  // Actualizar Arduino Cloud periódicamente
  updateCloud();

  // Control de ciclo rápido
  delay(80);  // ~12.5 Hz
}

//--------------------------------------------------------------------------
// Funciones profesionales
//--------------------------------------------------------------------------
```

```
void initCloudConnection() {
  Serial.println(">> Inicializando conexión Arduino Cloud...");

  // Configurar propiedades
  initProperties();

  // Conectar con manejo de errores profesional
  ArduinoCloud.begin(ArduinoIoTPreferredConnection);
  ArduinoCloud.setDebugMessageLevel(2);  // Nivel moderado de depuración

  Serial.print(">> Conectando");
  unsigned long cloudStart = millis();
  bool connectionLedState = false;

  while (!ArduinoCloud.connected()) {
    Serial.print(".");
    connectionLedState = !connectionLedState;
    digitalWrite(STATUS_LED, connectionLedState);
    delay(300);

    // Timeout después de 25 segundos
    if (millis() - cloudStart > 25000) {
      Serial.println("\n>> ERROR: Fallo conexión Cloud. Modo offline activado");
      cloudConnected = false;
      return;
    }
  }

  cloudConnected = true;
  digitalWrite(STATUS_LED, HIGH);
  Serial.println("\n>> Conexión Cloud establecida exitosamente!");
  ArduinoCloud.printDebugInfo();
}

void handleNetwork() {
  // Verificar WiFi cada 15 segundos
```

```
  static unsigned long lastWifiCheck = 0;
  if (millis() - lastWifiCheck > 15000) {
    if (WiFi.status() != WL_CONNECTED) {
      reconnectWiFi();
    }
    lastWifiCheck = millis();

    // Medir temperatura de la CPU
    cpuTemp = temperatureRead();
  }
}

void reconnectWiFi() {
  Serial.println(">> Reconectando a WiFi...");
  digitalWrite(STATUS_LED, LOW);
  WiFi.disconnect(true);
  delay(100);
  WiFi.begin(SSID, PASS);

  unsigned long wifiStart = millis();
  Serial.print(">> Conectando");

  while (WiFi.status() != WL_CONNECTED) {
    Serial.print(".");
    digitalWrite(STATUS_LED, millis() % 500 < 250);  // Blink rápido

    // Timeout después de 20 segundos
    if (millis() - wifiStart > 20000) {
      Serial.println("\n>> ERROR: No se pudo conectar al WiFi");
      return;
    }
    delay(300);
  }

  Serial.println("\n>> WiFi reconectado exitosamente!");
  Serial.printf(">> IP: %s\n", WiFi.localIP().toString().c_str());
```

```
  if (cloudConnected) {
    digitalWrite(STATUS_LED, HIGH);
  }
}

void readSensorData() {
  // Leer valor analógico
  int rawValue = analogRead(MQ_PIN);

  // Actualizar buffer de muestras
  rawReadings[readingIndex] = rawValue;
  readingIndex = (readingIndex + 1) % SAMPLES;
}

void updateCloud() {
  // Actualizar cada UPDATE_INTERVAL ms
  if (millis() - lastUpdate >= UPDATE_INTERVAL) {
    // Calcular promedio móvil
    MQ2 = calculateMovingAverage();
    lastUpdate = millis();

    // Actualizar Cloud
    ArduinoCloud.update();

    // Registrar estado del sistema
    logSystemStatus();
  }
}

int calculateMovingAverage() {
  long sum = 0;
  for (int i = 0; i < SAMPLES; i++) {
    sum += rawReadings[i];
  }
  return sum / SAMPLES;
```

```
}

void logSystemStatus() {
  // Calcular tiempo activo
  float uptime = (millis() - startTime) / 60000.0;  // Minutos

  // Obtener última lectura cruda
  int lastRaw = rawReadings[(readingIndex + SAMPLES - 1) % SAMPLES];

  Serial.println("\n--- ESTADO DEL SISTEMA ---");
  Serial.printf("Tiempo Activo: %.1f min\n", uptime);
  Serial.printf("Memoria Libre: %d bytes\n", ESP.getFreeHeap());
  Serial.printf("Temperatura CPU: %.1f°C\n", cpuTemp);
  Serial.printf("WiFi: %s | Cloud: %s\n",
          WiFi.status() == WL_CONNECTED ? "Conectado" : "Desconectado",
          cloudConnected ? "Conectado" : "Offline");
  Serial.printf("MQ2: RAW=%d | AVG=%d | Cloud=%d\n", lastRaw, MQ2, MQ2);
  Serial.println("-------------------------");
}
void onMQ2Change() {
  // Handler para cambios desde dashboard
                                          }
```

Código empleado para la programación en Python 3 para la creación de la interfaz GUI de los 2 Esp32 de Arduino Cloud

```
import customtkinter as ctk
import requests
import json
import threading
import time
from datetime import datetime, timedelta
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
import matplotlib.dates as mdates
```

```
import numpy as np
import pandas as pd
from PIL import Image
import os
import sys
import winsound
import openpyxl
from openpyxl.styles import Font, PatternFill, Border, Side, Alignment
from openpyxl.utils import get_column_letter

class FuturisticAirQualityApp(ctk.CTk):
  def __init__(self):
    super().__init__()

    # Configuración de la aplicación
    self.title("SISTEMA DE MONITOREO DE LA CALIDAD DEL AIRE SYNAPSE")
    self.geometry("1600x1000")
    self.state('zoomed')  # Maximizar ventana

    # Credenciales Arduino Cloud
    self.CLIENT_ID = "TU_CLIENT_ID"
    self.CLIENT_SECRET = "TU_CLIENT_SECRET"
    self.THING_ID_1 = "THING_ID_DISPOSITIVO_1"
    self.THING_ID_2 = "THING_ID_DISPOSITIVO_2"

    # Configuración de estilos futuristas
    ctk.set_appearance_mode("Dark")
    ctk.set_default_color_theme("blue")
    self.neon_blue = "#00c8ff"
    self.neon_green = "#39ff14"
    self.neon_purple = "#bc13fe"
    self.dark_bg = "#0d1117"
    self.card_bg = "#161b22"

    self.quality_widgets = {}
    # Variables de datos
```

```
self.device_data = {
  "device1": {
    "MQ1": 0,
    "last_update": "",
    "status": "Desconectado",
    "uptime": 0,
    "cpu_temp": 0,
    "memory": 0,
    "ip": ""
  },
  "device2": {
    "MQ2": 0,
    "last_update": "",
    "status": "Desconectado",
    "uptime": 0,
    "cpu_temp": 0,
    "memory": 0,
    "ip": ""
  }
}

# Historial de datos
self.historical_data = {
  "MQ1": {"timestamps": [], "values": []},
  "MQ2": {"timestamps": [], "values": []}
}

# Sistema de alertas
self.alerts = []
self.alert_rules = [
  {"name": "Crítico", "threshold": 2000, "color": "#ff0000", "sound": True},
  {"name": "Alto", "threshold": 1200, "color": "#ff6600", "sound": False},
  {"name": "Moderado", "threshold": 800, "color": "#ffff00", "sound": False},
  {"name": "Bajo", "threshold": 400, "color": "#00ff00", "sound": False}
]
```

```
        # Estadísticas
        self.stats = {
            "max_value": 0,
            "min_value": 9999,
            "avg_value": 0,
            "alert_count": 0
        }

        # Configurar cliente API
        self.access_token = ""
        self.token_expiration = 0
        self.setup_client()

        # Iniciar interfaz
        self.create_futuristic_ui()
        self.update_gui()

        # Iniciar hilos
        self.running = True
        threading.Thread(target=self.data_acquisition_thread, daemon=True).start()
        threading.Thread(target=self.update_historical_data, daemon=True).start()
        threading.Thread(target=self.check_alerts, daemon=True).start()

    def setup_client(self):
        """Configurar y autenticar el cliente de Arduino Cloud"""
        try:
            token_url = "https://api2.arduino.cc/iot/v1/clients/token"
            headers = {"Content-Type": "application/x-www-form-urlencoded"}
            data = {
                "grant_type": "client_credentials",
                "client_id": self.CLIENT_ID,
                "client_secret": self.CLIENT_SECRET,
                "audience": "https://api2.arduino.cc/iot"
            }

            response = requests.post(token_url, headers=headers, data=data)
```

```
            response.raise_for_status()

            token_data = response.json()
            self.access_token = token_data["access_token"]
            self.token_expiration = time.time() + token_data["expires_in"] - 300
            print("Autenticación exitosa con Arduino Cloud")
        except Exception as e:
            print(f"Error de autenticación: {str(e)}")
            self.access_token = ""

    def refresh_token(self):
        """Refrescar token si es necesario"""
        if time.time() >= self.token_expiration:
            print("Refrescando token...")
            self.setup_client()

    def get_device_data(self, thing_id):
        """Obtener datos de un dispositivo específico"""
        if not self.access_token:
            return {}

        try:
            url = f"https://api2.arduino.cc/iot/v2/things/{thing_id}/properties"
            headers = {
                "Authorization": f"Bearer {self.access_token}",
                "Content-Type": "application/json"
            }

            response = requests.get(url, headers=headers)
            response.raise_for_status()
            return response.json()
        except Exception as e:
            print(f"Error obteniendo datos: {str(e)}")
            return {}

    def process_device_data(self, device_key, thing_id, variable_name):
```

```
        """Procesar datos del dispositivo y actualizar estructura"""
        raw_data = self.get_device_data(thing_id)

        if not raw_data:
            self.device_data[device_key]["status"] = "Error conexión"
            return

        for item in raw_data:
            if item["name"] == variable_name:
                # Actualizar valor principal
                value = item["last_value"]
                self.device_data[device_key][variable_name] = value

                # Actualizar estadísticas
                if value > self.stats["max_value"]:
                    self.stats["max_value"] = value
                if value < self.stats["min_value"]:
                    self.stats["min_value"] = value

                # Actualizar metadatos
                self.device_data[device_key]["last_update"] = datetime.strptime(
                    item["updated_at"], "%Y-%m-%dT%H:%M:%S.%fZ"
                ).strftime("%d/%m/%Y %H:%M:%S")

                # Simular datos adicionales para demo
                self.device_data[device_key]["status"] = "Conectado"
                self.device_data[device_key]["uptime"] = round((time.time() - 3600) / 3600,
1)  # 1 hora
                self.device_data[device_key]["cpu_temp"] = round(40 +
np.random.uniform(0, 10), 1)
                self.device_data[device_key]["memory"] = round(20 + np.random.uniform(0,
10), 1)
                self.device_data[device_key]["ip"] = f"192.168.1.{np.random.randint(100,
200)}"

                break
```

```
    def data_acquisition_thread(self):
        """Hilo para adquisición continua de datos"""
        while self.running:
            try:
                self.refresh_token()

                # Obtener datos de ambos dispositivos
                self.process_device_data("device1", self.THING_ID_1, "MQ1")
                self.process_device_data("device2", self.THING_ID_2, "MQ2")

                # Calcular promedio
                values = [self.device_data["device1"]["MQ1"],
self.device_data["device2"]["MQ2"]]
                self.stats["avg_value"] = sum(values) / len(values) if values else 0

                # Actualizar GUI
                self.update_gui()

                time.sleep(3)
            except Exception as e:
                print(f"Error en hilo de adquisición: {str(e)}")
                time.sleep(10)

    def update_historical_data(self):
        """Actualizar datos históricos cada minuto"""
        while self.running:
            try:
                current_time = datetime.now()

                # Mantener solo datos de las últimas 24 horas
                for var in ["MQ1", "MQ2"]:
                    if len(self.historical_data[var]["timestamps"]) > 1440:
                        self.historical_data[var]["timestamps"] =
self.historical_data[var]["timestamps"][-1440:]
```

```
                self.historical_data[var]["values"] = self.historical_data[var]["values"][-
1440:]

            # Agregar nuevos datos
            if self.device_data["device1"]["MQ1"] is not None:
                self.historical_data["MQ1"]["timestamps"].append(current_time)

self.historical_data["MQ1"]["values"].append(self.device_data["device1"]["MQ1"])

            if self.device_data["device2"]["MQ2"] is not None:
                self.historical_data["MQ2"]["timestamps"].append(current_time)

self.historical_data["MQ2"]["values"].append(self.device_data["device2"]["MQ2"])

            # Actualizar gráficos y tabla
            self.update_plots()
            self.update_history_table()

            time.sleep(60)
        except Exception as e:
            print(f"Error en hilo histórico: {str(e)}")
            time.sleep(60)

    def check_alerts(self):
        """Verificar reglas de alerta continuamente"""
        while self.running:
            try:
                # Verificar alertas para ambos dispositivos
                for device_key, var_name in [("device1", "MQ1"), ("device2", "MQ2")]:
                    value = self.device_data[device_key].get(var_name, 0)
                    if value is None:
                        continue

                    device_name = "Estación 1" if device_key == "device1" else "Estación 2"

                    for rule in self.alert_rules:
```

```
                if value >= rule["threshold"]:
                    # Verificar si ya existe una alerta similar reciente
                    existing_alert = any(
                        a["device"] == device_key and
                        a["rule"] == rule["name"] and
                        (datetime.now() - a["timestamp"]).total_seconds() < 300
                        for a in self.alerts
                    )

                    if not existing_alert:
                        # Crear nueva alerta
                        alert = {
                            "id": len(self.alerts) + 1,
                            "device": device_key,
                            "device_name": device_name,
                            "variable": var_name,
                            "value": value,
                            "rule": rule["name"],
                            "threshold": rule["threshold"],
                            "timestamp": datetime.now(),
                            "acknowledged": False
                        }
                        self.alerts.append(alert)
                        self.stats["alert_count"] += 1

                        # Reproducir sonido si está configurado
                        if rule["sound"]:
                            threading.Thread(target=self.play_alert_sound).start()

                        # Actualizar interfaz de alertas
                        self.update_alerts_display()
                        break

            time.sleep(5)
        except Exception as e:
            print(f"Error en hilo de alertas: {str(e)}")
```

```
            time.sleep(10)

    def play_alert_sound(self):
        """Reproducir sonido de alerta"""
        try:
            for _ in range(3):
                winsound.Beep(800, 300)
                time.sleep(0.2)
        except:
            pass

    def create_futuristic_ui(self):
        """Crear interfaz de usuario futurista"""
        # Configurar grid principal
        self.grid_columnconfigure(0, weight=1)
        self.grid_rowconfigure(0, weight=1)

        # Frame principal con estilo de terminal futurista
        self.main_frame = ctk.CTkFrame(self, fg_color=self.dark_bg, corner_radius=0)
        self.main_frame.grid(row=0, column=0, sticky="nsew")
        self.main_frame.grid_columnconfigure(0, weight=1)
        self.main_frame.grid_rowconfigure(0, weight=1)

        # Crear pestañas
        self.tab_view = ctk.CTkTabview(self.main_frame, fg_color=self.card_bg,
                                       border_color=self.neon_blue, border_width=2,
                                       segmented_button_selected_color=self.neon_blue,
                                       segmented_button_selected_hover_color="#0099cc",
                                       text_color="white")
        self.tab_view.grid(row=0, column=0, padx=10, pady=10, sticky="nsew")

        # Pestañas
        self.tab_view.add("Dashboard")
        self.tab_view.add("Gráficos")
        self.tab_view.add("Dispositivos")
        self.tab_view.add("Alertas")
```

```
        self.tab_view.add("Estadísticas")
        self.tab_view.add("Histórico")

        # Configurar pestañas
        for tab in ["Dashboard", "Gráficos", "Dispositivos", "Alertas", "Estadísticas",
"Histórico"]:
            self.tab_view.tab(tab).grid_columnconfigure(0, weight=1)
            self.tab_view.tab(tab).grid_rowconfigure(0, weight=1)

        # Construir contenido de cada pestaña
        self.create_dashboard_tab()
        self.create_graphs_tab()
        self.create_devices_tab()
        self.create_alerts_tab()
        self.create_stats_tab()
        self.create_history_tab()

        # Barra de estado inferior
        self.status_bar = ctk.CTkFrame(self, height=30, fg_color="#000000")
        self.status_bar.grid(row=1, column=0, sticky="ew")

        self.status_label = ctk.CTkLabel(
            self.status_bar,
            text="Sistema inicializado | Conectando a Arduino Cloud...",
            text_color=self.neon_green,
            font=("Consolas", 12),
            anchor="w"
        )
        self.status_label.pack(side="left", padx=10, fill="x", expand=True)

        # Botones de control
        control_frame = ctk.CTkFrame(self.status_bar, fg_color="transparent")
        control_frame.pack(side="right", padx=10)

        ctk.CTkButton(
            control_frame,
```

```
            text="Exportar Excel",
            command=self.export_excel,
            width=120,
            fg_color="#1e6e1e",
            hover_color="#155015",
            border_color=self.neon_green,
            border_width=1,
            text_color="white",
            font=("Arial", 10, "bold")
        ).pack(side="left", padx=5)

        ctk.CTkButton(
            control_frame,
            text="Reiniciar",
            command=self.reset_stats,
            width=100,
            fg_color="#1a1a5e",
            hover_color="#121245",
            border_color=self.neon_blue,
            border_width=1,
            text_color="white",
            font=("Arial", 10, "bold")
        ).pack(side="left", padx=5)

        ctk.CTkButton(
            control_frame,
            text="Salir",
            command=self.quit_app,
            width=80,
            fg_color="#6e1e1e",
            hover_color="#501515",
            border_color="#ff5555",
            border_width=1,
            text_color="white",
            font=("Arial", 10, "bold")
        ).pack(side="left", padx=5)
```

```
def create_dashboard_tab(self):
    """Crear contenido de la pestaña Dashboard"""
    tab = self.tab_view.tab("Dashboard")
    tab.grid_columnconfigure(0, weight=1)
    tab.grid_rowconfigure(0, weight=1)

    # Frame principal del dashboard
    dashboard_frame = ctk.CTkFrame(tab, fg_color="transparent")
    dashboard_frame.grid(row=0, column=0, padx=10, pady=10, sticky="nsew")

    # Título futurista
    title_frame = ctk.CTkFrame(dashboard_frame, fg_color="transparent")
    title_frame.grid(row=0, column=0, columnspan=2, pady=(0, 20), sticky="ew")

    ctk.CTkLabel(
        title_frame,
        text="MONITOR DE CALIDAD DEL AIRE SYNAPSE",
        font=("Arial", 28, "bold"),
        text_color=self.neon_blue
    ).pack(side="left", padx=20)

    # Fecha y hora
    self.date_label = ctk.CTkLabel(
        title_frame,
        text=datetime.now().strftime("%d/%m/%Y %H:%M:%S"),
        font=("Arial", 16),
        text_color=self.neon_green
    )
    self.date_label.pack(side="right", padx=20)

    # Tarjetas de dispositivos
    devices_frame = ctk.CTkFrame(dashboard_frame, fg_color="transparent")
    devices_frame.grid(row=1, column=0, padx=10, pady=10, sticky="nsew")

    # Dispositivo 1
```

```
        self.create_device_card(devices_frame, "device1", "Estación MQ-135 #1", 0, 0)

        # Dispositivo 2
        self.create_device_card(devices_frame, "device2", "Estación MQ-135 #2", 0, 1)

        # Panel de estadísticas rápidas
        stats_frame = ctk.CTkFrame(dashboard_frame, fg_color=self.card_bg,
corner_radius=10)
        stats_frame.grid(row=2, column=0, padx=10, pady=10, sticky="ew")

        ctk.CTkLabel(
            stats_frame,
            text="ESTADÍSTICAS RÁPIDAS",
            font=("Arial", 16, "bold"),
            text_color=self.neon_purple
        ).grid(row=0, column=0, columnspan=4, pady=(10, 15))

        stats = [
            ("Máximo", "max_value"),
            ("Mínimo", "min_value"),
            ("Promedio", "avg_value"),
            ("Alertas", "alert_count")
        ]

        for i, (label, key) in enumerate(stats):
            ctk.CTkLabel(
                stats_frame,
                text=label + ":",
                font=("Arial", 12),
                text_color="#aaaaaa"
            ).grid(row=1, column=i, padx=10, pady=5, sticky="e")

            var = ctk.StringVar(value="0")
            setattr(self, f"stat_{key}_var", var)

            ctk.CTkLabel(
```

```
            stats_frame,
            textvariable=var,
            font=("Arial", 14, "bold"),
            text_color=self.neon_green
        ).grid(row=1, column=i+1, padx=(0, 20), pady=5, sticky="w")

    # Gráfico miniaturas
    mini_graph_frame = ctk.CTkFrame(dashboard_frame, fg_color=self.card_bg,
corner_radius=10)
    mini_graph_frame.grid(row=1, column=1, rowspan=2, padx=10, pady=10,
sticky="nsew")

    self.fig_mini, self.ax_mini = plt.subplots(figsize=(6, 4), facecolor=self.card_bg)
    self.ax_mini.tick_params(colors='white')
    self.ax_mini.set_facecolor(self.card_bg)

    self.canvas_mini = FigureCanvasTkAgg(self.fig_mini, master=mini_graph_frame)
    self.canvas_mini.get_tk_widget().pack(fill="both", expand=True, padx=10,
pady=10)

    ctk.CTkLabel(
        mini_graph_frame,
        text="TENDENCIA EN TIEMPO REAL",
        font=("Arial", 14, "bold"),
        text_color=self.neon_blue
    ).pack(pady=(10, 0))

  def create_device_card(self, parent, device_key, title, row, column):
    """Crear tarjeta futurista para un dispositivo"""
    card = ctk.CTkFrame(
        parent,
        fg_color=self.card_bg,
        corner_radius=15,
        border_width=2,
        border_color=self.neon_blue
    )
```

```
card.grid(row=row, column=column, padx=10, pady=10, sticky="nsew")

# Encabezado de tarjeta
header = ctk.CTkFrame(card, fg_color="transparent")
header.pack(fill="x", padx=15, pady=(15, 10))

ctk.CTkLabel(
    header,
    text=title,
    font=("Arial", 16, "bold"),
    text_color=self.neon_blue
).pack(side="left")

# Indicador de estado
status_frame = ctk.CTkFrame(header, fg_color="transparent")
status_frame.pack(side="right")

ctk.CTkLabel(
    status_frame,
    text="ESTADO:",
    font=("Arial", 12),
    text_color="#aaaaaa"
).pack(side="left", padx=(0, 5))

status_var = ctk.StringVar(value="Conectando...")
setattr(self, f"{device_key}_status_var", status_var)

ctk.CTkLabel(
    status_frame,
    textvariable=status_var,
    font=("Arial", 12, "bold"),
    text_color=self.neon_green
).pack(side="left")

# Valor principal con efecto neón
value_frame = ctk.CTkFrame(card, fg_color="#0a0a1a", corner_radius=10)
```

```
value_frame.pack(fill="x", padx=15, pady=10)

value_var = ctk.StringVar(value="0")
setattr(self, f"{device_key}_value_var", value_var)

ctk.CTkLabel(
    value_frame,
    textvariable=value_var,
    font=("Arial", 36, "bold"),
    text_color=self.neon_blue
).pack(side="left", padx=20, pady=10)

ctk.CTkLabel(
    value_frame,
    text="ppm",
    font=("Arial", 16),
    text_color=self.neon_green
).pack(side="left", pady=10)

# Indicador de calidad
quality_frame = ctk.CTkFrame(card, fg_color="transparent")
quality_frame.pack(fill="x", padx=15, pady=(5, 10))

ctk.CTkLabel(
    quality_frame,
    text="CALIDAD DEL AIRE:",
    font=("Arial", 12),
    text_color="#aaaaaa"
).pack(side="left", padx=(0, 10))

# Crear y almacenar las variables como atributos de instancia
setattr(self, f"{device_key}_quality_var", ctk.StringVar(value="--"))

self.quality_label = ctk.CTkLabel(
    quality_frame,
    textvariable=getattr(self, f"{device_key}_quality_var"),
```

```
            font=("Arial", 12, "bold"),
            text_color="white"
        )
        self.quality_label.pack(side="left")

        # Guardar referencia al label para actualizar color después
        self.quality_widgets[device_key] = self.quality_label

        self.quality_vars = {
            f"{device_key}_quality": ctk.StringVar(value="--"),
            f"{device_key}_quality_color": "white"  # Valor directo en lugar de StringVar
        }

        ctk.CTkLabel(
            quality_frame,
            textvariable=self.quality_vars[f"{device_key}_quality"],
            font=("Arial", 12, "bold"),
            text_color=self.quality_vars[f"{device_key}_quality_color"]  # Usar valor
directo
        ).pack(side="left")

        # Información adicional
        info_frame = ctk.CTkFrame(card, fg_color="transparent")
        info_frame.pack(fill="x", padx=15, pady=(0, 15))

        infos = [
            ("IP:", f"{device_key}_ip_var"),
            ("Últ. Actualización:", f"{device_key}_last_update_var"),
            ("Tiempo Activo:", f"{device_key}_uptime_var"),
            ("Temp CPU:", f"{device_key}_cpu_temp_var"),
            ("Memoria Libre:", f"{device_key}_memory_var")
        ]

        for i, (label, var_key) in enumerate(infos):
            row = i // 2
            col = i % 2 * 2
```

```
        ctk.CTkLabel(
            info_frame,
            text=label,
            font=("Arial", 11),
            text_color="#888888",
            width=120,
            anchor="e"
        ).grid(row=row, column=col, padx=(10, 2), pady=2, sticky="e")

        var = ctk.StringVar(value="--")
        setattr(self, var_key, var)

        ctk.CTkLabel(
            info_frame,
            textvariable=var,
            font=("Arial", 11),
            text_color="#dddddd",
            anchor="w"
        ).grid(row=row, column=col+1, padx=(0, 20), pady=2, sticky="w")

    def create_graphs_tab(self):
        """Crear pestaña de gráficos interactivos"""
        tab = self.tab_view.tab("Gráficos")
        tab.grid_columnconfigure(0, weight=1)
        tab.grid_rowconfigure(0, weight=1)

        # Frame para gráficos
        graph_frame = ctk.CTkFrame(tab, fg_color="transparent")
        graph_frame.grid(row=0, column=0, padx=10, pady=10, sticky="nsew")

        # Configuración global de estilos para matplotlib
        plt.style.use('dark_background')

        # Crear figura principal
        self.fig_main = plt.figure(figsize=(14, 8), facecolor=self.card_bg)
```

```
        gs = self.fig_main.add_gridspec(2, 2)

        self.ax_main1 = self.fig_main.add_subplot(gs[0, 0])
        self.ax_main2 = self.fig_main.add_subplot(gs[0, 1])
        self.ax_main3 = self.fig_main.add_subplot(gs[1, :])

        # Configurar estilos de gráficos con colores claros
        for ax in [self.ax_main1, self.ax_main2, self.ax_main3]:
            ax.set_facecolor(self.card_bg)

            # Configurar colores de texto
            ax.title.set_color('white')  # Color blanco para títulos
            ax.xaxis.label.set_color('white')  # Color blanco para etiqueta del eje X
            ax.yaxis.label.set_color('white')  # Color blanco para etiqueta del eje Y

            # Configurar colores de ticks
            ax.tick_params(axis='x', colors='white')
            ax.tick_params(axis='y', colors='white')

            # Configurar colores de bordes
            ax.spines['bottom'].set_color(self.neon_blue)
            ax.spines['top'].set_color(self.neon_blue)
            ax.spines['right'].set_color(self.neon_blue)
            ax.spines['left'].set_color(self.neon_blue)

        # Configurar títulos con estilo
        self.ax_main1.set_title('Estación 1: Tiempo Real', fontsize=12, color='white',
pad=20)
        self.ax_main2.set_title('Estación 2: Tiempo Real', fontsize=12, color='white',
pad=20)
        self.ax_main3.set_title('Comparación Histórica', fontsize=12, color='white',
pad=20)

        # Configurar grid
        for ax in [self.ax_main1, self.ax_main2, self.ax_main3]:
            ax.grid(True, linestyle='--', alpha=0.3, color='#444444')
```

```
        # Canvas para los gráficos
        self.canvas_main = FigureCanvasTkAgg(self.fig_main, master=graph_frame)
        self.canvas_main.get_tk_widget().pack(fill="both", expand=True, padx=10,
pady=10)

        # Controles de gráficos
        controls_frame = ctk.CTkFrame(graph_frame, fg_color="transparent")
        controls_frame.pack(fill="x", padx=10, pady=(0, 10))

        ctk.CTkLabel(
            controls_frame,
            text="Rango Temporal:",
            font=("Arial", 12),
            text_color="white"
        ).pack(side="left", padx=(10, 5))

        time_ranges = ["1 hora", "6 horas", "12 horas", "24 horas"]
        self.time_range_var = ctk.StringVar(value="6 horas")

        for tr in time_ranges:
            ctk.CTkRadioButton(
                controls_frame,
                text=tr,
                variable=self.time_range_var,
                value=tr,
                command=self.update_plots,
                font=("Arial", 11),
                text_color="white",
                fg_color=self.neon_blue,
                hover_color="#0099cc"
            ).pack(side="left", padx=10)

    def create_devices_tab(self):
        """Crear pestaña de detalles de dispositivos"""
        tab = self.tab_view.tab("Dispositivos")
```

```
        tab.grid_columnconfigure(0, weight=1)
        tab.grid_rowconfigure(0, weight=1)

        # Frame para dispositivos
        devices_frame = ctk.CTkFrame(tab, fg_color="transparent")
        devices_frame.grid(row=0, column=0, padx=10, pady=10, sticky="nsew")

        # Crear dos columnas
        devices_frame.grid_columnconfigure(0, weight=1)
        devices_frame.grid_columnconfigure(1, weight=1)
        devices_frame.grid_rowconfigure(0, weight=1)

        # Dispositivo 1
        self.create_device_detail(devices_frame, "device1", "Estación MQ-135 #1", 0, 0)

        # Dispositivo 2
        self.create_device_detail(devices_frame, "device2", "Estación MQ-135 #2", 0, 1)

    def create_device_detail(self, parent, device_key, title, row, column):
        """Crear panel de detalles para un dispositivo"""
        detail_frame = ctk.CTkFrame(
            parent,
            fg_color=self.card_bg,
            corner_radius=15,
            border_width=2,
            border_color=self.neon_blue
        )
        detail_frame.grid(row=row, column=column, padx=10, pady=10, sticky="nsew")

        # Título
        ctk.CTkLabel(
            detail_frame,
            text=title,
            font=("Arial", 18, "bold"),
            text_color=self.neon_blue
        ).pack(pady=(15, 10))
```

```
# Información técnica
tech_frame = ctk.CTkFrame(detail_frame, fg_color="transparent")
tech_frame.pack(fill="x", padx=20, pady=10)

tech_details = [
    ("ID Dispositivo:", "ESP32-001"),
    ("Firmware:", "v2.3.1"),
    ("Hardware:", "NodeMCU-32S"),
    ("Conectado desde:", "2025-06-23 08:45:12")
]

for label, value in tech_details:
    row_frame = ctk.CTkFrame(tech_frame, fg_color="transparent")
    row_frame.pack(fill="x", pady=2)

    ctk.CTkLabel(
        row_frame,
        text=label,
        font=("Arial", 12),
        text_color="#aaaaaa",
        width=150,
        anchor="w"
    ).pack(side="left")

    ctk.CTkLabel(
        row_frame,
        text=value,
        font=("Arial", 12),
        text_color="white"
    ).pack(side="left")

# Gráfico de rendimiento
perf_frame = ctk.CTkFrame(detail_frame, fg_color="#0a0a1a", corner_radius=10)
perf_frame.pack(fill="both", expand=True, padx=15, pady=15)
```

```
        ctk.CTkLabel(
            perf_frame,
            text="RENDIMIENTO DEL SISTEMA",
            font=("Arial", 14, "bold"),
            text_color=self.neon_purple
        ).pack(pady=(10, 5))

        # Gráfico ficticio de rendimiento
        fig_perf, ax_perf = plt.subplots(figsize=(6, 3), facecolor="#0a0a1a")
        ax_perf.set_facecolor("#0a0a1a")
        ax_perf.tick_params(colors='white')

        # Datos de ejemplo
        x = np.linspace(0, 10, 100)
        cpu_line, = ax_perf.plot(x, np.sin(x) * 0.5 + 0.5, 'c-', label='CPU')
        mem_line, = ax_perf.plot(x, np.cos(x) * 0.5 + 0.5, 'm-', label='Memoria')

        ax_perf.legend(facecolor=self.card_bg, labelcolor='white')
        ax_perf.set_ylim(0, 1)

        canvas_perf = FigureCanvasTkAgg(fig_perf, master=perf_frame)
        canvas_perf.draw()
        canvas_perf.get_tk_widget().pack(fill="both", expand=True, padx=10, pady=10)

    def create_alerts_tab(self):
        """Crear pestaña de sistema de alertas"""
        tab = self.tab_view.tab("Alertas")
        tab.grid_columnconfigure(0, weight=1)
        tab.grid_rowconfigure(0, weight=1)

        # Frame principal para alertas
        alerts_frame = ctk.CTkFrame(tab, fg_color="transparent")
        alerts_frame.grid(row=0, column=0, padx=10, pady=10, sticky="nsew")

        # Configurar columnas
        alerts_frame.grid_columnconfigure(0, weight=1)
```

```
        alerts_frame.grid_columnconfigure(1, weight=1)
        alerts_frame.grid_rowconfigure(1, weight=1)

        # Título y controles
        title_frame = ctk.CTkFrame(alerts_frame, fg_color="transparent")
        title_frame.grid(row=0, column=0, columnspan=2, sticky="ew", pady=(0, 10))

        ctk.CTkLabel(
            title_frame,
            text="SISTEMA DE ALERTAS",
            font=("Arial", 18, "bold"),
            text_color=self.neon_purple
        ).pack(side="left", padx=10)

        # Configuración de alertas
        config_frame = ctk.CTkFrame(alerts_frame, fg_color=self.card_bg,
corner_radius=10)
        config_frame.grid(row=1, column=0, padx=(0, 5), pady=5, sticky="nsew")

        ctk.CTkLabel(
            config_frame,
            text="CONFIGURACIÓN DE UMBRALES",
            font=("Arial", 14, "bold"),
            text_color=self.neon_blue
        ).pack(pady=(10, 15))

        # Reglas de alerta configurables
        for i, rule in enumerate(self.alert_rules):
            rule_frame = ctk.CTkFrame(config_frame, fg_color="#1a1a2e")
            rule_frame.pack(fill="x", padx=10, pady=5)

            # Color de alerta
            color_label = ctk.CTkLabel(
                rule_frame,
                text="●",
                text_color=rule["color"],
```

```
        font=("Arial", 16),
        width=30
    )
    color_label.pack(side="left", padx=(10, 5))

    # Nombre de alerta
    ctk.CTkLabel(
        rule_frame,
        text=rule["name"],
        font=("Arial", 12, "bold"),
        text_color=rule["color"],
        width=80
    ).pack(side="left", padx=5)

    # Umbral
    ctk.CTkLabel(
        rule_frame,
        text="Umbral:",
        font=("Arial", 11),
        text_color="#aaaaaa"
    ).pack(side="left", padx=(10, 0))

    threshold_var = ctk.IntVar(value=rule["threshold"])
    ctk.CTkEntry(
        rule_frame,
        textvariable=threshold_var,
        width=80,
        font=("Arial", 11),
        fg_color="#0d0d1a",
        border_color=rule["color"],
        border_width=1
    ).pack(side="left", padx=5)

    # Sonido
    sound_var = ctk.BooleanVar(value=rule["sound"])
    ctk.CTkCheckBox(
```

```
            rule_frame,
            text="Sonido",
            variable=sound_var,
            font=("Arial", 11),
            text_color="white",
            fg_color=rule["color"],
            hover_color=rule["color"]
        ).pack(side="left", padx=10)

        # Botón de guardar
        ctk.CTkButton(
            rule_frame,
            text="Guardar",
            width=70,
            font=("Arial", 10),
            fg_color="#1e3d1e",
            hover_color="#153015"
        ).pack(side="right", padx=10)

    # Lista de alertas activas
    alerts_list_frame = ctk.CTkFrame(alerts_frame, fg_color=self.card_bg,
corner_radius=10)
    alerts_list_frame.grid(row=1, column=1, padx=(5, 0), pady=5, sticky="nsew")

    ctk.CTkLabel(
        alerts_list_frame,
        text="ALERTAS ACTIVAS",
        font=("Arial", 14, "bold"),
        text_color=self.neon_blue
    ).pack(pady=(10, 15))

    # Canvas para lista de alertas con scrollbar
    alerts_canvas = ctk.CTkCanvas(
        alerts_list_frame,
        bg=self.card_bg,
        highlightthickness=0
```

```
        )
        alerts_canvas.pack(side="left", fill="both", expand=True, padx=10, pady=(0, 10))

        scrollbar = ctk.CTkScrollbar(
            alerts_list_frame,
            orientation="vertical",
            command=alerts_canvas.yview
        )
        scrollbar.pack(side="right", fill="y", padx=(0, 10), pady=(0, 10))

        alerts_canvas.configure(yscrollcommand=scrollbar.set)

        self.alerts_container = ctk.CTkFrame(alerts_canvas, fg_color=self.card_bg)
        self.alerts_container_id = alerts_canvas.create_window(
            (0, 0), window=self.alerts_container, anchor="nw")

        self.alerts_container.bind("<Configure>", lambda e: alerts_canvas.configure(
            scrollregion=alerts_canvas.bbox("all"))
        )

        # Configurar alertas iniciales
        self.update_alerts_display()

    def update_alerts_display(self):
        """Actualizar la visualización de alertas"""
        # Limpiar alertas existentes
        for widget in self.alerts_container.winfo_children():
            widget.destroy()

        # Ordenar alertas por fecha
        sorted_alerts = sorted(self.alerts, key=lambda x: x["timestamp"], reverse=True)

        # Mostrar alertas
        for alert in sorted_alerts:
            # Crear frame para alerta
            alert_frame = ctk.CTkFrame(
```

```
            self.alerts_container,
            fg_color="#1a1a2e",
            corner_radius=8,
            border_width=1,
            border_color=self.get_alert_color(alert["rule"])
        )
        alert_frame.pack(fill="x", padx=5, pady=5)

        # Encabezado de alerta
        header_frame = ctk.CTkFrame(alert_frame, fg_color="transparent")
        header_frame.pack(fill="x", padx=10, pady=(10, 5))

        ctk.CTkLabel(
            header_frame,
            text=f"ALERTA #{alert['id']} - {alert['rule'].upper()}",
            font=("Arial", 12, "bold"),
            text_color=self.get_alert_color(alert["rule"])
        ).pack(side="left")

        ctk.CTkLabel(
            header_frame,
            text=alert["timestamp"].strftime("%d/%m/%Y %H:%M:%S"),
            font=("Arial", 10),
            text_color="#aaaaaa"
        ).pack(side="right")

        # Detalles de alerta
        details_frame = ctk.CTkFrame(alert_frame, fg_color="transparent")
        details_frame.pack(fill="x", padx=10, pady=(0, 10))

        ctk.CTkLabel(
            details_frame,
            text=f"Dispositivo: {alert['device_name']}",
            font=("Arial", 11),
            text_color="#cccccc"
        ).grid(row=0, column=0, sticky="w")
```

```
        ctk.CTkLabel(
            details_frame,
            text=f"Variable: {alert['variable']}",
            font=("Arial", 11),
            text_color="#cccccc"
        ).grid(row=0, column=1, padx=(20, 0), sticky="w")

        ctk.CTkLabel(
            details_frame,
            text=f"Valor: {alert['value']} ppm",
            font=("Arial", 11, "bold"),
            text_color="#ffffff"
        ).grid(row=1, column=0, pady=(5, 0), sticky="w")

        ctk.CTkLabel(
            details_frame,
            text=f"Umbral: {alert['threshold']} ppm",
            font=("Arial", 11),
            text_color="#aaaaaa"
        ).grid(row=1, column=1, padx=(20, 0), pady=(5, 0), sticky="w")

        # Botones de acción
        action_frame = ctk.CTkFrame(alert_frame, fg_color="transparent")
        action_frame.pack(fill="x", padx=10, pady=(0, 10))

        if not alert["acknowledged"]:
            ctk.CTkButton(
                action_frame,
                text="Reconocer",
                width=100,
                font=("Arial", 10),
                fg_color="#1e3d1e",
                hover_color="#153015",
                command=lambda a=alert: self.acknowledge_alert(a)
            ).pack(side="left", padx=(0, 10))
```

```
        else:
            ctk.CTkLabel(
                action_frame,
                text="Reconocida",
                font=("Arial", 10, "bold"),
                text_color=self.neon_green
            ).pack(side="left", padx=(0, 10))

        ctk.CTkButton(
            action_frame,
            text="Ver Detalles",
            width=100,
            font=("Arial", 10),
            fg_color="#1a1a5e",
            hover_color="#121245"
        ).pack(side="left", padx=(0, 10))

        ctk.CTkButton(
            action_frame,
            text="Exportar",
            width=80,
            font=("Arial", 10),
            fg_color="#3d1a1a",
            hover_color="#301515"
        ).pack(side="right")

    def get_alert_color(self, alert_level):
        """Obtener color correspondiente al nivel de alerta"""
        for rule in self.alert_rules:
            if rule["name"] == alert_level:
                return rule["color"]
        return "#ffffff"

    def acknowledge_alert(self, alert):
        """Marcar una alerta como reconocida"""
        for a in self.alerts:
```

```
            if a["id"] == alert["id"]:
                a["acknowledged"] = True
                break
        self.update_alerts_display()

    def create_stats_tab(self):
        """Crear pestaña de estadísticas avanzadas"""
        tab = self.tab_view.tab("Estadísticas")
        tab.grid_columnconfigure(0, weight=1)
        tab.grid_rowconfigure(0, weight=1)

        # Frame para estadísticas
        stats_frame = ctk.CTkFrame(tab, fg_color="transparent")
        stats_frame.grid(row=0, column=0, padx=10, pady=10, sticky="nsew")

        # Gráficos de estadísticas
        fig_stats = plt.figure(figsize=(12, 8), facecolor=self.card_bg)
        gs = fig_stats.add_gridspec(2, 2)

        ax1 = fig_stats.add_subplot(gs[0, 0])
        ax2 = fig_stats.add_subplot(gs[0, 1])
        ax3 = fig_stats.add_subplot(gs[1, :])

        # Configurar estilos
        for ax in [ax1, ax2, ax3]:
            ax.set_facecolor(self.card_bg)
            ax.tick_params(colors='white')
            ax.spines['bottom'].set_color(self.neon_blue)
            ax.spines['top'].set_color(self.neon_blue)
            ax.spines['right'].set_color(self.neon_blue)
            ax.spines['left'].set_color(self.neon_blue)
            ax.title.set_color(self.neon_green)
            ax.xaxis.label.set_color('white')
            ax.yaxis.label.set_color('white')

        # Gráfico 1: Distribución de valores
```

```
        ax1.set_title('Distribución de Valores', fontsize=12)
        values = self.historical_data["MQ1"]["values"] +
self.historical_data["MQ2"]["values"]
        if values:
            ax1.hist(values, bins=20, color=self.neon_blue, edgecolor='black', alpha=0.7)
            ax1.set_xlabel('Valor (ppm)')
            ax1.set_ylabel('Frecuencia')

        # Gráfico 2: Comparación de estaciones
        ax2.set_title('Comparación de Estaciones', fontsize=12)
        if self.historical_data["MQ1"]["values"] and self.historical_data["MQ2"]["values"]:
            avg1 = sum(self.historical_data["MQ1"]["values"]) /
len(self.historical_data["MQ1"]["values"])
            avg2 = sum(self.historical_data["MQ2"]["values"]) /
len(self.historical_data["MQ2"]["values"])
            max1 = max(self.historical_data["MQ1"]["values"])
            max2 = max(self.historical_data["MQ2"]["values"])
            min1 = min(self.historical_data["MQ1"]["values"])
            min2 = min(self.historical_data["MQ2"]["values"])

            # Error bar corregido
            ax2.errorbar(
                ['Estación 1', 'Estación 2'],
                [avg1, avg2],
                yerr=[[avg1 - min1, avg2 - min2], [max1 - avg1, max2 - avg2]],
                fmt='o',
                color='red',
                capsize=5
            )
            ax2.set_ylabel('Promedio (ppm)')

        # Gráfico 3: Evolución temporal comparada
        ax3.set_title('Evolución Temporal Comparada', fontsize=12)
        if self.historical_data["MQ1"]["timestamps"] and
self.historical_data["MQ2"]["timestamps"]:
            ax3.plot(self.historical_data["MQ1"]["timestamps"],
```

```
                self.historical_data["MQ1"]["values"],
                self.neon_blue, label='Estación 1')
        ax3.plot(self.historical_data["MQ2"]["timestamps"],
                self.historical_data["MQ2"]["values"],
                self.neon_purple, label='Estación 2')
        ax3.set_ylabel('Valor (ppm)')
        ax3.legend(facecolor=self.card_bg, labelcolor='white')
        ax3.xaxis.set_major_formatter(mdates.DateFormatter('%d/%m %H:%M'))

    # Canvas para los gráficos
    canvas_stats = FigureCanvasTkAgg(fig_stats, master=stats_frame)
    canvas_stats.draw()
    canvas_stats.get_tk_widget().pack(fill="both", expand=True, padx=10, pady=10)

def create_history_tab(self):
    """Crear pestaña de datos históricos"""
    tab = self.tab_view.tab("Histórico")
    tab.grid_columnconfigure(0, weight=1)
    tab.grid_rowconfigure(0, weight=1)

    # Frame para histórico
    history_frame = ctk.CTkFrame(tab, fg_color="transparent")
    history_frame.grid(row=0, column=0, padx=10, pady=10, sticky="nsew")

    # Usar CTkScrollableFrame como contenedor
    scroll_frame = ctk.CTkScrollableFrame(history_frame)
    scroll_frame.pack(fill="both", expand=True, padx=10, pady=10)

    # Crear encabezados
    headers = ["Fecha", "Hora", "Estación 1 (ppm)", "Estación 2 (ppm)", "Diferencia"]
    for col, header in enumerate(headers):
        label = ctk.CTkLabel(
            scroll_frame,
            text=header,
            font=("Arial", 12, "bold"),
            text_color=self.neon_blue,
```

```
            width=120,
            anchor="center"
        )
        label.grid(row=0, column=col, padx=5, pady=5)

    # Variable para almacenar las filas de datos
    self.history_rows = []

    # Botones de exportación
    export_frame = ctk.CTkFrame(history_frame, fg_color="transparent")
    export_frame.pack(fill="x", padx=10, pady=(0, 10))

    ctk.CTkButton(
        export_frame,
        text="Exportar a Excel",
        command=self.export_excel,
        font=("Arial", 12),
        fg_color="#1e6e1e",
        hover_color="#155015",
        width=150
    ).pack(side="left", padx=10)

    ctk.CTkButton(
        export_frame,
        text="Exportar a CSV",
        command=self.export_csv,
        font=("Arial", 12),
        fg_color="#1a1a5e",
        hover_color="#121245",
        width=150
    ).pack(side="left", padx=10)

    ctk.CTkButton(
        export_frame,
        text="Generar Reporte",
        command=self.generate_report,
```

```
            font=("Arial", 12),
            fg_color="#5e1a5e",
            hover_color="#451545",
            width=150
        ).pack(side="right", padx=10)



    def update_history_table(self):
        """Actualizar la tabla de datos históricos"""
        # Limpiar filas existentes (excepto encabezados)
        for row in self.history_rows:
            for widget in row:
                widget.destroy()
        self.history_rows.clear()

        # Obtener los últimos 50 registros (o menos)
        num_records = min(50, len(self.historical_data["MQ1"]["timestamps"]))

        for i in range(num_records):
            row_data = []
            timestamp = self.historical_data["MQ1"]["timestamps"][-(i+1)]

            # Fecha y hora
            date_str = timestamp.strftime("%Y-%m-%d")
            time_str = timestamp.strftime("%H:%M:%S")

            # Valores de los sensores
            value1 = self.historical_data["MQ1"]["values"][-(i+1)]
            value2 = self.historical_data["MQ2"]["values"][-(i+1)] if i <
    len(self.historical_data["MQ2"]["values"]) else 0
            diff = abs(value1 - value2)

            # Crear widgets para la fila
            row_widgets = []
            for col, value in enumerate([date_str, time_str, value1, value2, diff]):
```

```
            widget = ctk.CTkLabel(
                scroll_frame,
                text=str(value),
                font=("Arial", 11),
                text_color="white",
                width=120,
                anchor="center"
            )
            widget.grid(row=i+1, column=col, padx=5, pady=2)
            row_widgets.append(widget)

        self.history_rows.append(row_widgets)









def update_history_table(self):
    """Actualizar la tabla de datos históricos"""
    # Limpiar filas existentes (excepto encabezados)
    for row in self.history_rows:
        for widget in row:
            widget.destroy()
    self.history_rows.clear()

    # Obtener los últimos 50 registros (o menos)
    num_records = min(50, len(self.historical_data["MQ1"]["timestamps"]))

    for i in range(num_records):
        row_data = []
```

```
            timestamp = self.historical_data["MQ1"]["timestamps"][-(i+1)]

            # Fecha y hora
            date_str = timestamp.strftime("%Y-%m-%d")
            time_str = timestamp.strftime("%H:%M:%S")

            # Valores de los sensores
            value1 = self.historical_data["MQ1"]["values"][-(i+1)]
            value2 = self.historical_data["MQ2"]["values"][-(i+1)] if i <
len(self.historical_data["MQ2"]["values"]) else 0
            diff = abs(value1 - value2)

            # Crear widgets para la fila
            row_widgets = []
            for col, value in enumerate([date_str, time_str, value1, value2, diff]):
                widget = ctk.CTkLabel(
                    scroll_frame,
                    text=str(value),
                    font=("Arial", 11),
                    text_color="white",
                    width=120,
                    anchor="center"
                )
                widget.grid(row=i+1, column=col, padx=5, pady=2)
                row_widgets.append(widget)

            self.history_rows.append(row_widgets)




    def update_gui(self):
        """Actualizar todos los elementos de la interfaz gráfica"""
        # Actualizar dispositivos
        self.update_device_ui("device1", "MQ1")
        self.update_device_ui("device2", "MQ2")
```

```
        # Actualizar estadísticas
        self.stat_max_value_var.set(f"{self.stats['max_value']} ppm")
        self.stat_min_value_var.set(f"{self.stats['min_value']} ppm")
        self.stat_avg_value_var.set(f"{self.stats['avg_value']:.1f} ppm")
        self.stat_alert_count_var.set(f"{self.stats['alert_count']}")

        # Actualizar fecha y hora
        self.date_label.configure(text=datetime.now().strftime("%d/%m/%Y
%H:%M:%S"))

        # Actualizar estado
        status_text = "Sistema Operativo | "
        status_text += f"Últ. Actualización: {datetime.now().strftime('%d/%m/%Y
%H:%M:%S')} | "
        status_text += f"Alertas Activas: {sum(1 for a in self.alerts if not
a['acknowledged'])}"
        self.status_label.configure(text=status_text)

        # Actualizar gráficos en miniatura
        self.update_mini_plot()

        # Actualizar cada segundo
        self.after(1000, self.update_gui)

    def update_device_ui(self, device_key, var_name):
        """Actualizar UI para un dispositivo específico"""
        data = self.device_data[device_key]

        # Verificar si las variables existen, si no, crearlas
        if not hasattr(self, f"{device_key}_quality_var"):
            setattr(self, f"{device_key}_quality_var", ctk.StringVar(value="--"))

        # Actualizar valor principal
        value = data.get(var_name, 0)
        if value is None:
```

```
            value = 0
        if hasattr(self, f"{device_key}_value_var"):
            getattr(self, f"{device_key}_value_var").set(f"{value}")

        # Determinar calidad del aire
        quality, color = self.determine_air_quality(value)
        if hasattr(self, f"{device_key}_quality_var"):
            getattr(self, f"{device_key}_quality_var").set(quality)

        # Actualizar el color del label de calidad si existe
        if device_key in self.quality_widgets:
            self.quality_widgets[device_key].configure(text_color=color)

        # Actualizar información de estado (con verificaciones)
        if hasattr(self, f"{device_key}_status_var"):
            getattr(self, f"{device_key}_status_var").set(data.get("status", "Desconocido"))
        if hasattr(self, f"{device_key}_last_update_var"):
            getattr(self, f"{device_key}_last_update_var").set(data.get("last_update", "--"))
        if hasattr(self, f"{device_key}_uptime_var"):
            getattr(self, f"{device_key}_uptime_var").set(f"{data.get('uptime', 0):.1f}
horas")
        if hasattr(self, f"{device_key}_cpu_temp_var"):
            getattr(self, f"{device_key}_cpu_temp_var").set(f"{data.get('cpu_temp', 0):.1f}
°C")
        if hasattr(self, f"{device_key}_memory_var"):
            getattr(self, f"{device_key}_memory_var").set(f"{data.get('memory', 0):.1f}%
libre")
        if hasattr(self, f"{device_key}_ip_var"):
            getattr(self, f"{device_key}_ip_var").set(data.get("ip", "192.168.1.XXX"))

    def determine_air_quality(self, value):
        """Determinar calidad del aire basado en valor del sensor"""
        if value < 300:
            return "Excelente", "#39ff14"  # Neon Green
        elif 300 <= value < 600:
            return "Buena", "#00c8ff"     # Neon Blue
```

```
        elif 600 <= value < 900:
            return "Moderada", "#ffcc00"   # Yellow
        elif 900 <= value < 1200:
            return "Pobre", "#ff6600"      # Orange
        else:
            return "Peligrosa", "#ff0000"  # Red

    def update_plots(self):
        """Actualizar gráficos principales"""
        if not self.historical_data["MQ1"]["timestamps"]:
            return

        # Determinar rango temporal
        time_range = self.time_range_var.get()
        hours = 1 if time_range == "1 hora" else 6 if time_range == "6 horas" else 12 if
time_range == "12 horas" else 24
        time_limit = datetime.now() - timedelta(hours=hours)

        # Filtrar datos según el rango temporal
        filtered_data = {"MQ1": {"timestamps": [], "values": []}, "MQ2": {"timestamps": [],
"values": []}}

        for i, ts in enumerate(self.historical_data["MQ1"]["timestamps"]):
            if ts >= time_limit:
                filtered_data["MQ1"]["timestamps"].append(ts)

filtered_data["MQ1"]["values"].append(self.historical_data["MQ1"]["values"][i])

        for i, ts in enumerate(self.historical_data["MQ2"]["timestamps"]):
            if ts >= time_limit:
                filtered_data["MQ2"]["timestamps"].append(ts)

filtered_data["MQ2"]["values"].append(self.historical_data["MQ2"]["values"][i])

        # Actualizar gráfico 1 (Estación 1)
        self.ax_main1.clear()
```

```
        if filtered_data["MQ1"]["timestamps"]:
            self.ax_main1.plot(
                filtered_data["MQ1"]["timestamps"],
                filtered_data["MQ1"]["values"],
                color=self.neon_blue,
                linewidth=2
            )
        self.ax_main1.set_title('Estación 1: Tiempo Real', fontsize=12, color='white')
        self.ax_main1.set_ylabel('Concentración (ppm)', fontsize=10, color='white')
        self.ax_main1.grid(True, linestyle='--', alpha=0.3)
        self.ax_main1.tick_params(axis='x', colors='white')
        self.ax_main1.tick_params(axis='y', colors='white')

        # Actualizar gráfico 2 (Estación 2)
        self.ax_main2.clear()
        if filtered_data["MQ2"]["timestamps"]:
            self.ax_main2.plot(
                filtered_data["MQ2"]["timestamps"],
                filtered_data["MQ2"]["values"],
                color=self.neon_purple,
                linewidth=2
            )
        self.ax_main2.set_title('Estación 2: Tiempo Real', fontsize=12, color='white')
        self.ax_main2.set_ylabel('Concentración (ppm)', fontsize=10, color='white')
        self.ax_main2.grid(True, linestyle='--', alpha=0.3)
        self.ax_main2.tick_params(axis='x', colors='white')
        self.ax_main2.tick_params(axis='y', colors='white')

        # Actualizar gráfico 3 (Comparación)
        self.ax_main3.clear()
        if filtered_data["MQ1"]["timestamps"]:
            self.ax_main3.plot(
                filtered_data["MQ1"]["timestamps"],
                filtered_data["MQ1"]["values"],
                color=self.neon_blue,
                linewidth=1.5,
```

```
            label='Estación 1'
        )
    if filtered_data["MQ2"]["timestamps"]:
        self.ax_main3.plot(
            filtered_data["MQ2"]["timestamps"],
            filtered_data["MQ2"]["values"],
            color=self.neon_purple,
            linewidth=1.5,
            label='Estación 2'
        )
    self.ax_main3.set_title('Comparación Histórica', fontsize=12, color='white')
    self.ax_main3.set_ylabel('Concentración (ppm)', fontsize=10, color='white')
    self.ax_main3.set_xlabel('Tiempo', fontsize=10, color='white')
    self.ax_main3.legend(facecolor=self.card_bg, labelcolor='white')
    self.ax_main3.grid(True, linestyle='--', alpha=0.3)
    self.ax_main3.tick_params(axis='x', colors='white')
    self.ax_main3.tick_params(axis='y', colors='white')

    # Formato de fechas
    self.ax_main3.xaxis.set_major_formatter(mdates.DateFormatter('%H:%M'))
    self.ax_main3.xaxis.set_major_locator(mdates.HourLocator(interval=max(1,
hours//6)))
    self.fig_main.autofmt_xdate()

    self.canvas_main.draw()

  def update_mini_plot(self):
    """Actualizar gráfico en miniatura del dashboard"""
    if not self.historical_data["MQ1"]["timestamps"]:
        return

    self.ax_mini.clear()

    # Limitar a las últimas 50 muestras
    limit = min(50, len(self.historical_data["MQ1"]["timestamps"]))
    timestamps = self.historical_data["MQ1"]["timestamps"][-limit:]
```

```
        values1 = self.historical_data["MQ1"]["values"][-limit:]
        values2 = self.historical_data["MQ2"]["values"][-limit:]

        if timestamps and values1:
            self.ax_mini.plot(timestamps, values1, self.neon_blue, label='Estación 1')
        if timestamps and values2:
            self.ax_mini.plot(timestamps, values2, self.neon_purple, label='Estación 2')

        self.ax_mini.set_facecolor(self.card_bg)
        self.ax_mini.tick_params(colors='white')
        self.ax_mini.xaxis.set_major_formatter(mdates.DateFormatter('%H:%M'))
        self.ax_mini.grid(True, linestyle='--', alpha=0.3, color='#444444')

        # Añadir leyenda
        if values1 or values2:
            self.ax_mini.legend(loc='upper right', facecolor=self.card_bg,
labelcolor='white')

        self.canvas_mini.draw()

    def export_excel(self):
        """Exportar datos a Excel con formato profesional"""
        try:
            # Crear DataFrame con datos históricos
            timestamps = self.historical_data["MQ1"]["timestamps"]
            data = {
                "Fecha": [ts.strftime("%Y-%m-%d") for ts in timestamps],
                "Hora": [ts.strftime("%H:%M:%S") for ts in timestamps],
                "Estacion1_ppm": self.historical_data["MQ1"]["values"],
                "Estacion2_ppm": self.historical_data["MQ2"]["values"],
                "Diferencia_ppm": [
                    abs(v1 - v2) for v1, v2 in zip(
                        self.historical_data["MQ1"]["values"],
                        self.historical_data["MQ2"]["values"]
                    )
                ]
```

```
        }
        df = pd.DataFrame(data)

        # Generar nombre de archivo
        timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
        filename = f"air_quality_report_{timestamp}.xlsx"

        # Crear libro de Excel con formato profesional
        workbook = openpyxl.Workbook()
        sheet = workbook.active
        sheet.title = "Datos Calidad del Aire"

        # Encabezados
        headers = ["Fecha", "Hora", "Estación 1 (ppm)", "Estación 2 (ppm)",
"Diferencia (ppm)"]
        for col_num, header in enumerate(headers, 1):
            cell = sheet.cell(row=1, column=col_num, value=header)
            cell.font = Font(bold=True, color="FFFFFF")
            cell.fill = PatternFill(start_color="0070C0", end_color="0070C0",
fill_type="solid")
            cell.alignment = Alignment(horizontal="center")

        # Datos
        for row_num, row_data in enumerate(df.values, 2):
            for col_num, value in enumerate(row_data, 1):
                sheet.cell(row=row_num, column=col_num, value=value)

        # Formato condicional para diferencias
        red_fill = PatternFill(start_color="FFC7CE", end_color="FFC7CE",
fill_type="solid")
        yellow_fill = PatternFill(start_color="FFEB9C", end_color="FFEB9C",
fill_type="solid")

        for row in sheet.iter_rows(min_row=2, min_col=5, max_col=5):
            for cell in row:
                if cell.value > 300:
```

```
                cell.fill = red_fill
            elif cell.value > 150:
                cell.fill = yellow_fill

    # Ajustar anchos de columna
    for col in sheet.columns:
        max_length = 0
        column = get_column_letter(col[0].column)
        for cell in col:
            try:
                if len(str(cell.value)) > max_length:
                    max_length = len(str(cell.value))
            except:
                pass
        adjusted_width = (max_length + 2) * 1.2
        sheet.column_dimensions[column].width = adjusted_width

    # Guardar archivo
    workbook.save(filename)
    self.status_label.configure(text=f"Reporte exportado a {filename}")
  except Exception as e:
    self.status_label.configure(text=f"Error exportando: {str(e)}")

def export_csv(self):
  """Exportar datos a CSV"""
  try:
    # Crear DataFrame
    timestamps = self.historical_data["MQ1"]["timestamps"]
    data = {
      "Fecha": [ts.strftime("%Y-%m-%d") for ts in timestamps],
      "Hora": [ts.strftime("%H:%M:%S") for ts in timestamps],
      "Estacion1_ppm": self.historical_data["MQ1"]["values"],
      "Estacion2_ppm": self.historical_data["MQ2"]["values"],
      "Diferencia_ppm": [
        abs(v1 - v2) for v1, v2 in zip(
          self.historical_data["MQ1"]["values"],
```

```
                self.historical_data["MQ2"]["values"]
            )
        ]
    }
    df = pd.DataFrame(data)

    # Generar nombre de archivo
    timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
    filename = f"air_quality_data_{timestamp}.csv"

    # Exportar a CSV
    df.to_csv(filename, index=False)
    self.status_label.configure(text=f"Datos exportados a {filename}")
  except Exception as e:
    self.status_label.configure(text=f"Error exportando: {str(e)}")

def generate_report(self):
  """Generar reporte estadístico avanzado"""
  try:
    # Generar nombre de archivo
    timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
    filename = f"air_quality_analysis_{timestamp}.txt"

    # Calcular estadísticas
    all_values = self.historical_data["MQ1"]["values"] +
self.historical_data["MQ2"]["values"]
    stats = {
      "Muestras Totales": len(all_values),
      "Máximo Global": max(all_values) if all_values else 0,
      "Mínimo Global": min(all_values) if all_values else 0,
      "Promedio Global": sum(all_values) / len(all_values) if all_values else 0,
      "Alertas Registradas": self.stats["alert_count"],
      "Primera Muestra":
self.historical_data["MQ1"]["timestamps"][0].strftime("%Y-%m-%d %H:%M") if
self.historical_data["MQ1"]["timestamps"] else "N/A",
```

```
            "Última Muestra": self.historical_data["MQ1"]["timestamps"][-
1].strftime("%Y-%m-%d %H:%M") if self.historical_data["MQ1"]["timestamps"] else
"N/A"
        }

        # Crear reporte
        with open(filename, "w") as f:
            f.write("REPORTE DE CALIDAD DEL AIRE\n")
            f.write("============================\n\n")
            f.write(f"Generado el: {datetime.now().strftime('%Y-%m-%d
%H:%M:%S')}\n\n")

            f.write("ESTADÍSTICAS GLOBALES\n")
            f.write("----------------------\n")
            for key, value in stats.items():
                f.write(f"{key}: {value}\n")

            f.write("\nRESUMEN POR ESTACIÓN\n")
            f.write("--------------------\n")
            for i, station in enumerate(["Estación 1", "Estación 2"], 1):
                key = "MQ1" if i == 1 else "MQ2"
                values = self.historical_data[key]["values"]
                if values:
                    f.write(f"\n{station}:\n")
                    f.write(f"  Muestras: {len(values)}\n")
                    f.write(f"  Máximo: {max(values)}\n")
                    f.write(f"  Mínimo: {min(values)}\n")
                    f.write(f"  Promedio: {sum(values)/len(values):.2f}\n")

            f.write("\nALERTAS REGISTRADAS\n")
            f.write("-------------------\n")
            for alert in self.alerts:
                f.write(f"[Alerta #{alert['id']}] {alert['device_name']} - {alert['value']} ppm
({alert['rule']})\n")
                f.write(f"  Fecha: {alert['timestamp'].strftime('%Y-%m-%d %H:%M')}\n")
```

```
                f.write(f"  Estado: {'Reconocida' if alert['acknowledged'] else
'Pendiente'}\n\n")

            self.status_label.configure(text=f"Reporte generado: {filename}")
        except Exception as e:
            self.status_label.configure(text=f"Error generando reporte: {str(e)}")

    def reset_stats(self):
        """Reiniciar estadísticas del sistema"""
        self.stats = {
            "max_value": 0,
            "min_value": 9999,
            "avg_value": 0,
            "alert_count": 0
        }
        self.alerts = []
        self.update_alerts_display()
        self.status_label.configure(text="Estadísticas reiniciadas")

    def quit_app(self):
        """Salir de la aplicación de manera controlada"""
        self.running = False
        time.sleep(1)
        self.destroy()

if __name__ == "__main__":
    app = FuturisticAirQualityApp()
                                        app.mainloop()
```

DECLARACIÓN Y AUTORIZACIÓN

Yo, **Jerez Ronquillo, Diego Antonio** con C.C: 0926942517 autor del Trabajo de Integración Curricular: **Supervisión de la calidad de aire en ambiente cerrado usando tecnología embebida,** previo a la obtención del título de **Ingeniero eléctrico mecánico con mención en gestión empresarial,** en la Universidad Católica de Santiago de Guayaquil.

1.- Declaro tener pleno conocimiento de la obligación que tienen las instituciones de educación superior, de conformidad con el Artículo 144 de la Ley Orgánica de Educación Superior, de entregar a la SENESCYT en formato digital una copia del referido trabajo de integración curricular para que sea integrado al Sistema Nacional de Información de la Educación Superior del Ecuador para su difusión pública respetando los derechos de autor.

2.- Autorizo a la SENESCYT a tener una copia del referido trabajo de integración curricular, con el propósito de generar un repositorio que democratice la información, respetando las políticas de propiedad intelectual vigentes.

Guayaquil, 01 de septiembre del 2025

_____

Jerez Ronquillo, Diego Antonio

C.C: 0926942517

REPOSITORIO NACIONAL EN CIENCIA Y TECNOLOGÍA			
FICHA DE REGISTRO DE TESIS/TRABAJO DE TITULACIÓN			
TEMA Y SUBTEMA:	**Supervisión de la calidad de aire en ambiente cerrado usando tecnología embebida.**		
AUTOR(ES)	**Jerez Ronquillo, Diego Antonio**		
REVISOR(ES)/TUTOR(ES)	**Ing. Bohórquez Escobar, Celso Bayardo. PhD.**		
INSTITUCIÓN:	**Universidad Católica de Santiago de Guayaquil**		
FACULTAD:	**Facultad de Educación Técnica para el Desarrollo**		
CARRERA:	**Ingeniería en Eléctrico Mecánico**		
TITULO OBTENIDO:	**Ingeniero Eléctrico Mecánico con mención en Gestión Empresarial**		
FECHA DE PUBLICACIÓN:	**01 de septiembre del 2025**	**No. DE PÁGINAS:**	**122**
ÁREAS TEMÁTICAS:	**Automatización, Controladores, Sensor de monitoreo**		
PALABRAS CLAVES/ KEYWORDS:	**ESP32, Arduino Cloud, MQTT, MQ-135, IoT, Python.**		

RESUMEN/ABSTRACT

El presente trabajo de integración curricular se sitúa en el diseño de un sistema de una red de nodos sensores MQ-135 controlados por microcomputadoras ESP32, conectados a la nube de Arduino y apoyados por una aplicación en Python, dejará la monitorización en vivo de los indicadores más importantes sobre la calidad del aire en un taller automotriz de Guayaquil. El objetivo central se basa en analizar la calidad de aire en un ambiente cerrado usando tablas de datos estándares nacionales e internacionales que permitan generar alarmas de peligro, si algún parámetro se sale de las normativas. Se desarrolla un sistema eléctrico, electrónico que permita medir los parámetros para medir la calidad de aire. La metodología se hará usando un método combinado. En la etapa primera, se llevará a cabo un examen detallado de libros y reglas sobre calidad del aire, sensores MQ-135, microcontroladores IoT y espacios en la nube. Después, se creará la red de sensores: se elegirán lugares importantes dentro del taller, se ajustarán los MQ-135 con modelos de gases y se programarán los ESP32 para tener y mandar datos por medio de MQTT.

ADJUNTO PDF:	☒ SI	☐ NO
CONTACTO CON AUTOR/ES:	**Teléfono:**	**E-mail: diego.jerez@cu.ucsg.edu.ec**
CONTACTO CON LA INSTITUCIÓN (C00RDINADOR DEL PROCESO UTE)::	**Nombre: Ricardo Xavier Ubilla González**	
	Teléfono: +593999528515	
	E-mail: Ricardo.ubilla@cu.ucsg.edu.ec	

SECCIÓN PARA USO DE BIBLIOTECA	
Nº. DE REGISTRO (en base a datos):	
Nº. DE CLASIFICACIÓN:	
DIRECCIÓN URL (tesis en la web):	