Universidad Internacional de La Rioja

Escuela de Ingeniería y Tecnología

Máster Universitario en Dirección e Ingeniería de Sitios Web

# Usos del IoT: Mejora de la accesibilidad en el hogar



| | |
|---|---|
| Trabajo fin de estudio presentado por: | Abel Soler Calatayud |
| Tipo de trabajo: | Propuesta de arquitectura |
| Rama (Profesional/Investigación): | Profesional |
| Director/a: | Oscar Sanjuán Martínez |

*"Dedicat a la meua dona Vicen per la seua paciència i també per al pròxim projecte en comú que s'anomenarà Pau."*

*"A mi director de TFM por sus consejos y guías que me han ayudado a avanzar en este proyecto."*

# Resumen

El envenenamiento accidental por exposición a gases y vapores es la causa principal de fallecimiento, superando incluso las intoxicaciones autoinfligidas, dentro del grupo de las intoxicaciones por gases. Estos fallecimientos, en más de la mitad de las ocasiones, se producen dentro del hogar. Además, las estadísticas indican que la mayor parte de ellas se producen en personas mayores a 50 años. La intoxicación por monóxido de carbono (CO) es una emergencia médica común y su diagnóstico médico es complicado de determinar porque las sintomatologías son poco específicas (cefalea, náuseas, mareo, arritmias cardíacas, etc.).

El objetivo de este proyecto consiste en estudiar y proponer una solución de arquitectura IoT usando Arduino para construir un sistema de detección de gases en el hogar. Este sistema será capaz de registrar los datos medidos usando sensores que sirvan para aplicaciones en interiores y de bajo coste. Posteriormente, estos datos serán visualizados a través de una aplicación para dispositivos móviles Android y además mostrará alertas cuando detecte niveles altos de gas. Esta arquitectura estará constituida por microcontroladores ESP32 y ESP8266, sensores de detección de gases como el MQ-135 y el uso de APP Inventor para desarrollar la aplicación Android.

**Palabras clave:** IoT, ESP32, ESP8266, Smart Home, detección de gases

# Abstract

Accidental poisoning due to exposure to gases and vapors is the leading cause of death, surpassing even self-inflicted poisoning, within the group of gas intoxications. More than half of these deaths occurs at home. In addition, statistics indicate that most of them are produced in people over the age 50. Carbon monoxide (CO) poisoning is a common medical emergency and its medical diagnosis is complicated to determine because the symptoms are not very specific (headache, nausea, dizziness, cardiac arrhythmias, etc.).

The purpose of this project is to study and design an IoT architecture solution using Arduino to build a home gas detection system. This system will be able to collect measured data using sensors that are suitable for indoor and low-cost applications. Subsequently, this data will be visualized through an application for Android mobile devices and will also display alerts when it detects high gas levels. This architecture will be composed of ESP32 and ESP8266 microcontrollers, gas detection sensors such as the MQ-135 and the use of APP Inventor to develop the Android application.

**Keywords**: IoT, ESP32, ESP8266, Smart Home, gas detection

# Índice de contenidos

# Índice de figuras

# Índice de tablas

# 1. Introducción

Lo que empezó como un título de una presentación PowerPoint en 1999, hoy en día es una realidad. El concepto de internet de las cosas o *Internet of Things* (IoT) está cada vez más presente en nuestras vidas. Kevin Ashton, padre del internet de las cosas, creó este concepto para explicar un proyecto que tenía en mente, poner etiquetas de identificación de radiofrecuencia (RFID) y demás sensores en los productos de la cadena de suministro, de este modo disponer de una trazabilidad del producto.

Pero no fue hasta 2008 cuando este término empezó a utilizarse de manera significativa, tal vez un efecto derivado por la explosión de Twitter lo que ayudó a expandir esta palabra (#IoT) dentro de los mensajes de 140 caracteres de esta plataforma. En 2011, tras la publicación de un estudio de Gartner en el que incluía el IoT dentro de la lista de las nuevas tecnologías emergentes, fue cuando se terminó de consolidar este concepto. Maayan (2020) resume la tendencia ascendente de la evolución y uso del IoT en los últimos años:

- En 2018 había 7 billones de dispositivos IoT.
- En 2019 el número activo de dispositivos IoT alcanzan los 26’6 billones.
- Durante el 2020 los expertos estiman que se instalarán 31 billones de dispositivos IoT.
- Cada segundo, 127 nuevos dispositivos IoT se conectan a la web.
- En 2021 la previsión será de 35 billones de dispositivos IoT instalados.
- En 2025 más de 75 billones de dispositivos IoT estarán conectados en la red.

Actualmente los dispositivos IoT forman parte de nuestra sociedad en forma de wearables, sensores de temperatura y humedad, control de iluminación exterior,… que facilitan nuestro día a día recopilando información de manera que podamos realizar acciones concretas en base a los datos almacenados. Esta recopilación constante de datos sobre cualquier entorno abre un amplio abanico de posibilidades en cuanto a la realización de proyectos como pueden ser: de prevención de accidentes, mejora de accesibilidad en el hogar, monitorización de nuestra salud, sistemas de vigilancia, etc.

En este trabajo fin de máster se pretende aprovechar la capacidad de los distintos dispositivos IoT que hay en el mercado para plantear e implementar una solución a un problema concreto. De manera más específica, se planteará una solución para prevenir accidentes domésticos como pueden ser la inhalación de gases nocivos para la salud como es el monóxido de carbono (CO) producido por malas combustiones de elementos domésticos tales como calefactores, estufas, chimeneas, calderas,…

## 1.1. Justificación del Trabajo

Según el artículo elaborado por Bartolomé, M. T., Amores, P., Cuesta, E. y Gallego, N. (2010), la intoxicación aguda por monóxido de carbono (CO) es una emergencia médica común y una causa frecuente de muerte intencional o accidental. Además su diagnóstico o reconocimiento por parte del cuerpo médico es complicado debido a las características propias de este gas. También sus síntomas son tan poco específicos que añaden más complejidad para la detección de la intoxicación (problemas respiratorios, náuseas y vómitos, dolor torácico, coma, mareo, somnolencia, cefalea, arritmias cardiacas, etc.).

**Tabla 1. Posibles fuentes de exposición a monóxido de carbono ambiental.**

| | Total (%) |
|---|---|
| Vivienda en área de alta densidad de tráfico. | 24 (47,1) |
| Uso de chimenea, horno de aceite, quemador de gas o leña para calentar el domicilio. | 27 (52,9) |
| Uso de aparatos de combustión sin ventilación (calentadores, cocinas, generador eléctrico) en interiores. | 10 (19,6) |
| Reparación reciente de horno, calderas y/o chimenea. | 7 (13,7) |
| Olvido de cocina de gas y/o carbón encendida. | 5 (9,8) |
| Realiza regularmente barbacoas. | 5 (9,8) |
| Trabajos en taller de reparación de coches. | 4 (7,8) |
| Olvido de coche encendido en el garaje. | 0 (0) |
| Permanencia en lugares con salida de ventilación bloqueada. | 0 (0) |

Fuente: Buchelli, y otros, 2014.

Asimismo, se tomará como motivación principal para la consecución de este trabajo las conclusiones obtenidas de los estudios como el de Buchelli y otros (2014), Zaragozano, J. F., Estupiñá, C. F., Espatolero, P. A., Dufol, A. F., y López, J. O. (2005) y Robles, M. D. M., Carreira, J. M. F., Suárez, I. G., Diéguez, O. F., y Torres, G. R. (2009) donde corroboran que estas

intoxicaciones se producen la mayor parte en el domicilio particular de la persona afectada y que la intoxicación por CO representa la causa más común de lesiones y muerte a nivel mundial (Thom, 2002).

Tal y como se ha comentado en la introducción, una de las características que tienen los dispositivos IoT es la capacidad de obtención de datos para ser posteriormente tratados. Teniendo en cuenta esta característica y enlazándolo con la problemática detectada por la intoxicación por CO, podemos elaborar el marco base sobre el que se desarrollará este proyecto.

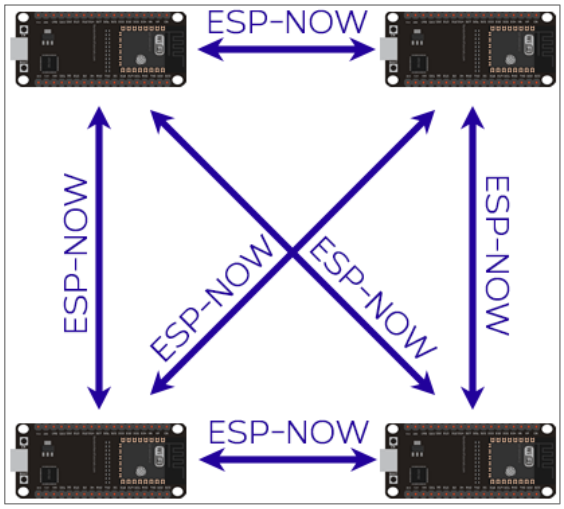
## 1.2. Planteamiento de la solución

Con ello, la solución que se plantea en este trabajo es la monitorización y control de la calidad del aire en los espacios cerrados (normalmente hogares) a través de una aplicación para dispositivos móviles usando componentes IoT. Esta aplicación tendrá la suficiente autonomía para notificar al usuario cuando ésta detecta que los niveles de calidad del aire, captados por los sensores, están descendiendo hasta umbrales perjudiciales para la salud. De esta manera se conseguirá evitar intoxicaciones por inhalación de CO.

Estos dispositivos IoT se comunicarán entre sí usando el protocolo de comunicación ESP-NOW (figura 1) y se comunicarán con la aplicación instalada en el dispositivo móvil (smartphone o Tablet) usando la tecnología Wi-Fi. Además, como punto diferenciador del resto de soluciones del mercado, esta propuesta de arquitectura no requiere de conectividad a internet ni elementos de comunicaciones intermedios tales como el router, hub o centralita.

De este modo, se pretende conseguir llegar a los hogares dónde el uso de la tecnología está bastante restringido, bien sea por no disponer de recursos económicos o formen parte de un colectivo vulnerable como las personas mayores o de avanzada edad. Solamente se necesita un dispositivo con sistema operativo Android capaz de conectarse a una red Wi-Fi.

***Figura 1.*** *Conectividad bidireccional con ESP-NOW.*




Fuente: Elaboración propia, 2021.

Los componentes electrónicos encargados de la orquestación de toda la comunicación, captación de datos de sensores y tratamiento de los datos serán los ESP32 y ESP8266. Aunque en esencia pueden ser lo mismo (el ESP32 es una evolución del ESP8266) ambos representan a la misma familia de chips SoC (System on a Chip) de bajo costo creados y desarrollados por Espressif Systems[1].

Estos chips tendrán una tarea concreta, por una parte el ESP32 será el encargado de la comunicación con la aplicación y el chip ESP8266 será el orquestador de la recogida del dato del sensor, enviarlo al ESP32 y también comunicarse con otros chips ESP8266 usando el protocolo ESP-NOW recogiendo los datos de los otros sensores. La comunicación entre ambos chips se realizará por los puertos serie RX y TX disponibles a través de un cable sobre el que se enviarán los datos usando protocolo de comunicación implementado ad-hoc para este proyecto.

[1] Espressif Systems es una empresa china privada, de tipo fabless, del sector de diseño electrónico de semiconductores creada en 2008.

En cuanto a la captura de datos, los sensores encargados serán los de la familia MQ. Existen una gran variedad de sensores dentro del grupo donde cada uno está destinado a identificar un gas concreto, no todos los sensores MQ pueden detectar el mismo tipo de gas. En este trabajo se utilizará el MQ-135 que es capaz de medir los niveles de gases peligrosos como son los del tipo NOx, amoníaco (NH3), benceno, alcohol, humo y niveles de CO.

## 1.3. Estructura de la Memoria

En el primer apartado de este trabajo correspondiente al estado del arte se hablará de la propia definición o concepto del IoT, además se establecerá el contexto de aplicación de esta solución y se enumerarán las distintas alternativas que existen actualmente en el mercado y trabajos existentes similares que han servido como base de ideas para llevar a cabo este proyecto.

Asimismo se profundizará en el análisis de los chips ESP32 y ESP8266 que forman el fundamento de la arquitectura en la que se desarrollará este trabajo y el software utilizado para la programación de los mismos será Arduino.cc. En cuanto al desarrollo de la aplicación móvil, se enumerarán distintos frameworks de trabajo y se detallará el IDE utilizado en este proyecto, concretamente App Inventor.

Siguiendo con la estructura de la memoria, el siguiente apartado se corresponde con los objetivos (generales y específicos) y la metodología utilizada. Referente al objetivo general de este trabajo lo podemos definir como el uso del IoT para la mejora de la accesibilidad en el hogar. Esta mejora de la accesibilidad es entendida desde el punto de vista del incremento de la comodidad de la persona convirtiendo el hogar en un sitio seguro en el que vivir.

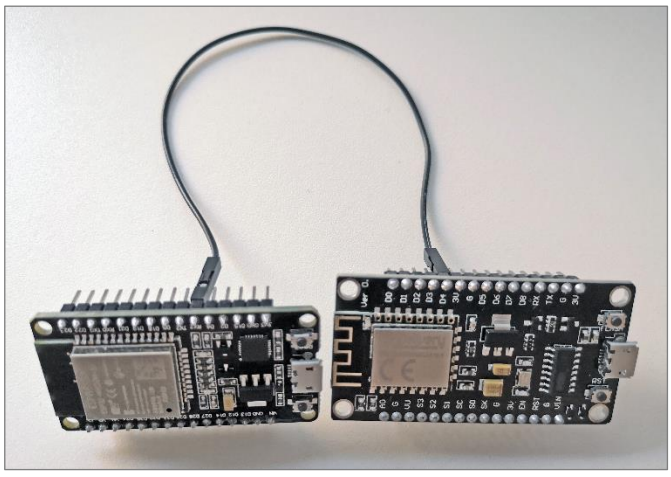
Para alcanzar este objetivo se crearán cinco objetivos más específicos como son la creación de una APP que se comunique con los dispositivos ESP32, montar un servidor web en el ESP32 capaz de enviar y recibir peticiones HTTP, establecer la comunicación entre el ESP32 y el ESP8266 a través del puerto serie usando un protocolo de intercambio de

información ad-hoc., enlazar los distintos ESP8266 entre sí para que puedan enviar y recibir información simultáneamente y finalmente implementar la comunicación entre el ESP8266 y el sensor de gas MQ-135.

Referente a la metodología de trabajo a seguir, la idea ha sido la implementación de cada uno de los objetivos específicos de manera atómica de menor a mayor complejidad. Una vez alcanzado dichos objetivos, se enlazarán los distintos desarrollos hasta alcanzar el objetivo principal.

Como primer paso se comprobará cómo el dispositivo ESP8266 es capaz de establecer e interpretar los datos del sensor MQ-135. Esta conexión se realizará mediante un cable al pin de entrada analógica que dispone la placa. Por otro lado, tras la consecución del objetivo de conexión entre los distintos ESP8266 a través del protocolo ESP-NOW, se comprobará que se envían y reciben los datos de los distintos sensores correctamente. Seguidamente se implementará un protocolo de comunicación entre el ESP32 y el ESP8266 a través de las conexiones de sus puertos series de envío y recepción de datos (figura 2).

***Figura 2.*** *Conexión física entre el ESP32 y el ESP8266.*



Fuente: Elaboración propia, 2021.

Una vez conseguidos estos tres objetivos específicos, el siguiente paso será devolver la información recopilada cuando el ESP32 reciba una petición HTTP a una URL específica. Finalmente la APP se encargará de realizar las peticiones HTTP hacia el servidor del ESP32, interpretar la respuesta recibida y representarla gráficamente. Esta aplicación también generará una notificación push cuando alguno de los sensores exceda de un umbral definido.

Continuando con el siguiente apartado, el desarrollo específico de la contribución, ahondará sobre cada uno de los objetivos específicos definidos en el apartado anterior, llegando a un nivel de detalle como por ejemplo centrar qué tipo de tecnología se utilizará en la comunicación HTTP, cómo se definirá el protocolo de comunicación entre placas, uso de extensiones para App Inventor, etc.

El siguiente apartado se corresponderá con la evaluación dónde se explicará cómo se desarrollará la solución de arquitectura y la evaluación de su funcionamiento, llegando incluso a probar la detección de alguno de los gases que es capaz de interpretar el sensor MQ-135 como es la presencia elevada de los niveles de alcohol en aire.

Para finalizar, la memoria se cerrará con el capítulo de conclusiones y futuras líneas de trabajo. En este apartado se explicará si finalmente la arquitectura planteada y aplicada podría valer como posible solución a la necesidad detectada en el objetivo general.

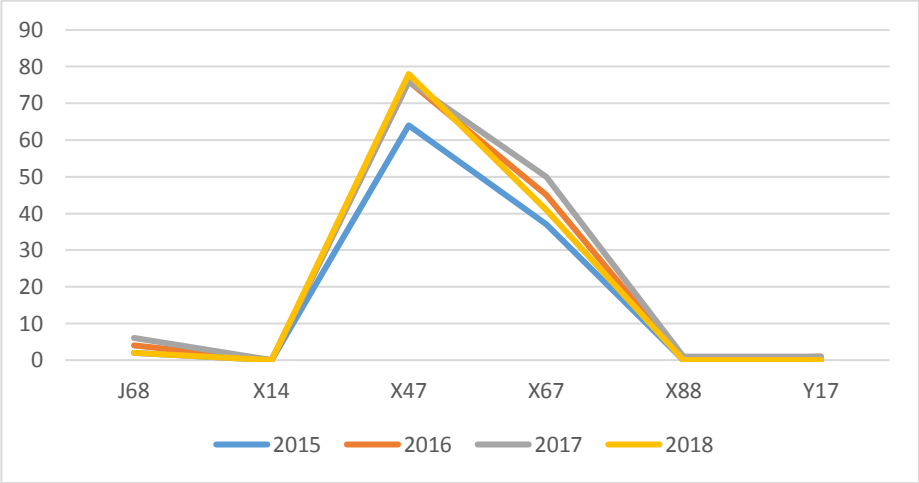
# 2. Contexto y estado del arte

## 2.1. Contexto

Según los últimos datos disponibles del Instituto Nacional de Estadística (en adelante INE), entre 2015 y 2018 hubo un total de 484 muertes relacionadas con la intoxicación de gases. Este dato representa un fallecimiento por cada tres días durante estos cuatro años. Desgranando estos datos, existen diversas causas tales como afecciones respiratorias debidas a inhalación de gases, humos, vapores y sustancias químicas (J68), contacto con aire y gases

calientes (X14), envenenamiento accidental por (exposición a) otros gases y vapores (X47), envenenamiento autoinfligido intencionalmente por (exposición a) otros gases y vapores (X67), agresión con gases y vapores (X88) y envenenamiento por (exposición a) otros gases y vapores, de intención no determinada (Y17).

***Figura 3.*** *Fallecimientos por inhalación de gases.*



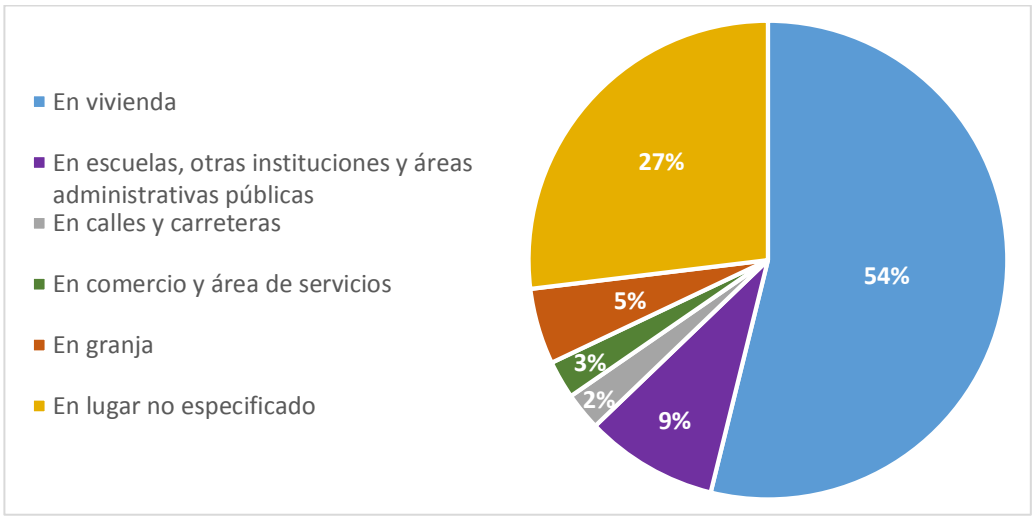

Fuente: INE, 2021.

Como se puede apreciar en la figura 3, los motivos principales por fallecimiento durante ese periodo de tiempo son el envenenamiento accidental por (exposición a) otros gases y vapores (X47) y el envenenamiento autoinfligido intencionalmente por (exposición a) otros gases y vapores (X67). A partir del 2019 el INE añade la descripción del monóxido de carbono (CO) modificando así la clasificación de estos dos grupos, envenenamiento accidental por (exposición a) monóxido de carbono y otros gases y vapores y el envenenamiento autoinfligido intencionalmente por (exposición a) monóxido de carbono y otros gases y vapores.

De estos dos grupos predominantes, el caso accidental de intoxicación (X47) es el grupo sobre el que centraremos la base de desarrollo de este proyecto. Si desglosamos los lugares dónde se han producido estos envenenamientos, el INE indica que en 2018 el 54% de los casos tienen lugar dentro del hogar (figura 4). Estas cifras están en línea con la conclusión del estudio

de Thom (2002) que indicaba el domicilio particular de la persona afectada y la intoxicación por CO representa la causa más común de lesiones y muerte a nivel mundial.

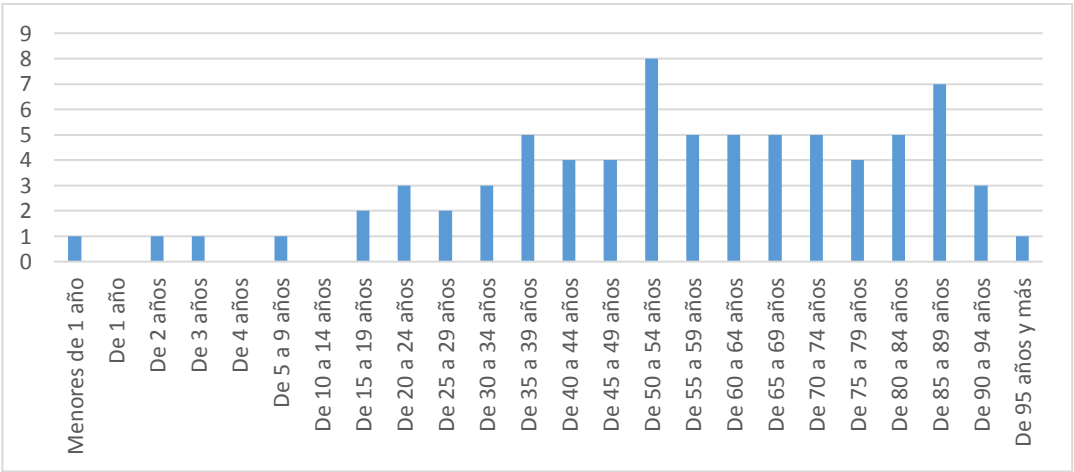
***Figura 4.*** *Grupos de zonas por intoxicación de gases en 2018.*




Fuente: INE, 2021.

Si además también se analiza el rango de edad sobre el que se producen estas intoxicaciones, en la figura 5 se ilustra, por edad, la media de fallecimientos desde el año 2015 al año 2018. En ella podemos observar que la mayor parte de los envenenamientos se producen en edades superiores a los 50 años.

***Figura 5.*** *Grupos de edad afectados por intoxicación entre 2015 y 2018.*




Fuente: INE, 2021.

Intentar reducir la cifra de fallecimientos por envenenamiento accidental por CO presenta un reto en cuanto a la detección temprana de emisiones de estos gases, evitando de este modo, que sean inhalados de manera inconsciente por la persona y que termina provocando finalmente su muerte.

Hoy en día existen múltiples dispositivos que controlan y avisan cuando detectan una presencia elevada de CO en el aire. En cuanto a la conectividad, también existe una gran variedad de opciones como por ejemplo ZigBee, Z-Wave, Wi-Fi y NB-IoT, pero todas ellas necesitan de un hub o concentrador y/o conectividad a internet para que la APP del teléfono móvil pueda funcionar.

***Figura 6.*** *Detector de monóxido de carbono HS2CA-NB.*


Fuente: Heiman, 2021.

## 2.2. Definición del IoT

Si intentamos buscar una definición única en cuanto al concepto de IoT, no es trivial. Podemos entender el IoT como un sistema interrelacionado de objetos o dispositivos conectados a internet. La idea de IoT abarca desde máquinas mecánicas y digitales, a objetos, animales o personas que se identifican de manera única (UID) en la red IoT y con la capacidad de transferir datos a través de dicha red sin requerir interacción de persona a persona y de persona a ordenador. Dicha red, puede estar conectada a internet mediante un concentrador, centralita, router, etc. De este modo, la tecnología IoT permite que los sistemas, dispositivos

y usuarios conectados a dicha red, puedan crear una amplia red incrementando la posibilidad de conectividad entre todos los dispositivos que la componen.

En 2011, en un simposio celebrado en la Feria de Hannover (Alemania) entre expertos en industria e informática, se debatió sobre el futuro y la evolución de lo que conocemos hoy en día como "la Industria 4.0". El tema central estaba definido y era reducir el trabajo manual para incrementar el trabajo intelectual. Con este cambio de paradigma en la industria pretendían iniciar un proceso de digitalización en las empresas en el que no bastaba simplemente en crear máquinas que se limitasen a producir, sino ir un paso más allá, máquinas que sean capaces de aprender y conectarlas entre sí, llegando de este modo al concepto más próximo del internet de las cosas.

El entorno industrial no es el único ámbito sobre el que los dispositivos IoT tienen aplicación, Maayan (2020) en su artículo, propone una clasificación en cinco grupos sobre las principales aplicaciones del internet de las cosas:

1- IoT del consumidor: electrodomésticos, iluminación, sensores, asistencia por voz, wearables, etc.

2- IoT comercial: aplicaciones IoT en las industrias de salud y transportes como marcapasos inteligentes, sistemas de monitorización y comunicación de flotas, etc.

3- IoT industrial (IIoT): sistemas de control digital, evaluación estadística, agricultura inteligente, Big Data industrial, gestión de activos en el almacén o trazabilidad de un pedido…

4- IoT de infraestructura: conectividad de ciudades inteligentes mediante el uso de sensores de infraestructura, sistemas de gestión y aplicaciones de usuario fáciles de usar.

5- IoT militar (IoMT): aplicación de tecnologías de IoT en el campo militar, como robots para vigilancia y biometría de uso humano para el combate.

## 2.3. Áreas de aplicación del IoT

Aunque la clasificación de Maayan (2020) establecía los cinco grupos, el uso o ámbito de aplicación del internet de las cosas no tiene límites. El único límite que puede existir es la creatividad, ingenio y capacidad de la persona que plantea y desarrolla la solución.

### 2.3.1. Industria

Como se comentaba anteriormente, en la industria es uno de los ámbitos de uso donde puede promover grandes cambios en cuanto a los modelos de producción incorporando el IoT. Algunos ámbitos de uso podían ser:

- Favorecer el proceso de almacenaje de productos: Si durante el inventariado de los productos en una empresa, cada producto dispone de un sistema de conexión a la red, podría realizarse una trazabilidad del producto indicando la ubicación exacta, cantidad disponible, información sobre el entorno como la temperatura, etc. Estos parámetros usados de una manera correcta podrían repercutir favorablemente en los tiempos de búsqueda ya que se podrían automatizar procesos.

- Mejora en el mantenimiento de la cadena de producción: Colocando dispositivos en puntos críticos de la cadena de producción, podría centralizarse el diagnóstico de incidencias y tener una visión global del estado de la cadena de producción. De este modo mejoraría el proceso de mantenimiento y seguimiento reduciendo así la intervención humana en tareas de control.

### 2.3.2. Ciudades

Otro ámbito muy favorable para la integración de dispositivos IoT es en las ciudades o las comúnmente llamadas Smart City, algunas de las aplicaciones podrían ser:

- Gestión automatizada de los servicios: Usar los distintos dispositivos IoT para control de temperatura, analizar en tiempo real el entorno y determinar ciertas acciones como encendido del alumbrado, riego de parques o control de la basura podrían reducir considerablemente el tiempo dedicado por las personas para la realización de dichas tareas.

- Monitorización del tráfico y aparcamiento: La gestión de los semáforos midiendo y tomando decisiones en tiempo real sobre el flujo del tráfico, podría reducir los niveles de $CO_2$ causados por los atascos y accidentes. También, la implementación de un sistema de control sobre plazas de aparcamiento disponibles ayudaría a optimizar el gasto de combustible de los usuarios mientras buscan un lugar para aparcar.

### 2.3.3. Hogar

El ámbito doméstico puede ser uno de los que más está avanzando con el uso de dispositivos IoT. Aunque explícitamente el usuario no percibe el uso, de manera implícita existen infinidad de electrodomésticos que son IoT.

- Electrodomésticos inteligentes: Estos dispositivos son una mezcla del electrodoméstico tradicional pero con un dispositivo IoT incorporado que conectado a un router o concentrador puede realizar sugerencias al usuario, obtener información de internet (como por ejemplo recetas de cocina) e incluso solicitar productos agotados en el frigorífico.

- Control del estado de la casa: Colocando dispositivos con sensores de temperatura, sensores de humedad, sensores de gas, sensores de humo,… puede ayudar a realizar acciones en función de la información recopilada por estos dispositivos como por ejemplo la detección precoz un incendio, detectar una fuga de agua, etc.

- Seguridad en el hogar: En muchos casos, los sistemas de alarma de hoy en día integran dispositivos IoT dotando de la capacidad de informar en tiempo real al usuario sobre el estado de los sensores de movimiento, visualización de vídeo a

través de una cámara, sensores de apertura de puertas, etc. ofreciendo confort y tranquilidad para las personas que habitan en la casa.

### 2.3.4. Salud

Actualmente, con la combinación de los smartphone y los wearables como por ejemplo los smartwatch, podemos disponer de cierta información que nos pueden ayudar a mejorar nuestro estilo de vida.

- Monitorización de nuestra salud: Utilizando sensores de control de pulsaciones, niveles de oxígeno en sangre, control de las fases del sueño, etc. pueden ser utilizados por nosotros mismos o también por un equipo médico especializado. Esta recopilación de datos podría ayudar a detectar carencias o posibles problemas en nuestro estilo de vida.

- Control de personas dependientes: Personas con algún tipo de discapacidad o personas de avanzada edad que necesiten de un control diario por una persona, usando este tipo de sensores podría monitorizarse de manera remota el estado de salud y actividad del individuo. Esto mejoraría sustancialmente el seguimiento y detección de problemas derivados en este tipo de personas como pueden ser las bajadas de tensión, caídas fortuitas, monitorización del ritmo cardíaco, etc.

Después de enumerar algunos posibles usos del internet de las cosas, se puede apreciar que el ámbito de aplicación es extenso y que incluso en algunos casos ya forman parte en nuestro día a día. Por este motivo es necesario acotar el ámbito del contexto sobre el que se desarrolla este proyecto. De manera más específica este trabajo utilizará el IoT en el ámbito del hogar y la salud con el uso sensores para la monitorización de la calidad del aire.

## 2.4. Protocolos de comunicación en IoT

En cuanto al diseño de la solución IoT, hay algunas características clave que deben ser consideradas, como por ejemplo, los recursos limitados de los sensores inalámbricos, el

entorno de la red sobre la que están montados y cómo se realizará el tratamiento de datos masivos recopilados por dichos sensores (Sheng, Mahapatra, Zhu, & Leung, 2015). En este sentido, uno de los principales aspectos a tener en cuenta es el protocolo de comunicación. Como se puede observar en la tabla 2, existen distintos protocolos de comunicación en que los distintos dispositivos IoT pueden crear esa pequeña red distribuida en base a las necesidades del proyecto.

**Tabla 2. Mapeo de aplicaciones y tecnologías radio de corto alcance.**

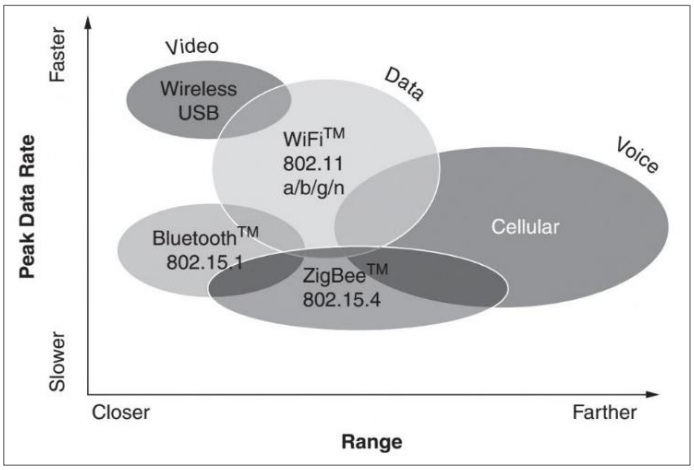
| | Technical Summary | Typical Radio Band | Transmission Range | Data Rate | Applications |
|---|---|---|---|---|---|
| Wi-Fi | It is probably the most widely used Wireless local area network (WLAN) technology based on the IEEE 802.11 series of standards. | 2.4 GHz,<br>5 GHz | 150 m | 54 Mbps | Video and monitoring based applications, smart home. |
| Bluetooth | Bluetooth low energy technology is a global standard, which enables devices with coin cell batteries to be wirelessly connected to standard Bluetooth enabled devices and services. | 2.4 GHz (v1.x, v4),<br>5 GHz (v3.0) | 10 – 150 m | 1 Mbps,<br>24 Mbps | Remote access, Sports & Fitness, Indoor positioning (HAIP), smartphone based applications. |
| ZigBee (IEEE 802.15.4) | A well-defined protocol stack for WSN with features of self-deployment, low complexity, low data rate and low cost, etc., based on IEEE 802.15.4 standards. | 780 MHz (China),<br>868 MHz,<br>915 MHz,<br>2.4 GHz | 100 – 300 m | 20 Kbps,<br>40 Kbps,<br>250 Kbps | Smart Energy, Home Automation, Building Automation, Health care, Remote Control, Retail Services. |
| RFID | A fast developing radio technology used to transfer data from an electronic tac, which includes identification, information collection, etc. | 125 KHz (LF),<br>13.56 MHz (HF),<br>433 MHz (UHF),<br>2.4 GHz (MW) | <10 cm,<br><1 m,<br>4 – 20 m,<br>60 – 100 m | 1 – 5 Kbps,<br>6.62 – 26.48 Kbps,<br>40 – 640 Kbps,<br>200 – 400 Kbps | Logistic, E-car license, one pass card. |
| 433MHz enabled proprietary solutions | Proprietary solutions by using one of the most commonly used ISM (industrial, scientific and medical) radio band in China. | 433 MHz | 300 – 1500 m | <10 Kbps | Home security, environment monitoring. |

Fuente: Sheng, Mahapatra, Zhu, & Leung, 2015.

### 2.4.1. ZigBee

ZigBee fue concebido por la ZigBee Alliance el año 1990 y no fue lanzado hasta el 2014. Este protocolo de redes inalámbricas está definido bajo la norma IEEE 802.15.4. Como bien apunta Gislason (2008) en su libro, mientras otros estándares inalámbricos están centrados en añadir más y más funcionalidades, ZigBee está orientado al uso de microcontroladores de 8bits, encargados de mandar pequeños paquetes de datos de sensores de temperatura, controlar una luz, etc. Estas características sitúan a ZigBee dentro de un mercado dónde las otras tecnologías inalámbricas no están diseñadas para ello (figura 7).

***Figura 7.*** *Características de las distintas tecnologías inalámbricas.*




Fuente: Gislason, 2008.

Gracias a la baja potencia de transmisión y un rango moderado, los dispositivos ZigBee pueden llegar a funcionar durante años, en cambio, para otro tipo de tecnologías la autonomía con batería podríamos estar hablando de horas o quizás días (Gislason, 2008). La cuota de mercado que intenta abarcar ZigBee se denomina “control y redes de sensores inalámbricos”, o tal vez de manera más simple "control inalámbrico". De hecho, el lema de ZigBee es “Wireless Control That Simply Works”.

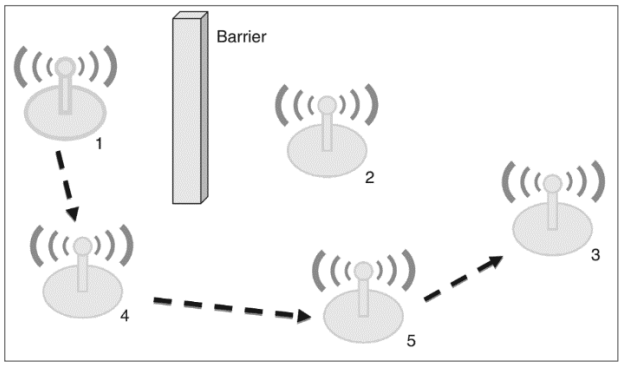
Este protocolo dispone de unas necesidades únicas que dentro del mercado de control inalámbrico convierte a ZigBee como un gran candidato. ZigBee es muy fiable, económico, altamente seguro (seguridad AES de 128 bit para cifrado y autenticación), capaz de lograr una

potencia muy baja e implementado sobre un estándar global abierto. Este protocolo, para lograr los criterios de bajos coste y bajo consumo, se estableció una restricción, una tasa baja de datos.

Otra característica diferenciadora de la red ZigBee respecto a las redes inalámbricas convencionales es la red de malla. La red de malla presenta tres propiedades, gran rango de cobertura a través de múltiples saltos, creación de la red ad-hoc y el descubrimiento automático de rutas y reparación automática de dichas rutas.

En la figura 8 podemos observar un ejemplo de red de malla ZigBee donde el nodo 1 necesita comunicarse con el nodo 3 pero no tienen visibilidad directa entre estos nodos. El camino más corto o con el menor número de saltos sería usando el nodo número 2, pero como existe una barrera que impide la visión directa entre el nodo 1 y el nodo 2, la adaptabilidad y el autodescubrimiento de rutas, la red se adapta y comprueba que pasando a través de los nodos 4 y 5 puede alcanzar el nodo 3 y comunicarse con él.

***Figura 8.*** *Adaptabilidad de la red ZigBee.*




Fuente: Gislason, 2008.

### 2.4.2. Z-Wave

Z-Wave es otro protocolo de comunicación inalámbrica desarrollado por Sigma Designs, Inc. que proporciona cifrado de paquetes, protección de integridad y servicios de autenticación de dispositivos. Z-Wave está ganando impulso frente al protocolo ZigBee con

respecto a la automatización del hogar (Fouladi & Ghanoun, 2013). Este protocolo, respecto ZigBee, está menos sujeto a interferencias de la señal porque no opera sobre la banda de 2,4 Ghz, altamente poblada y compartida por dispositivos Bluetooth y Wi-Fi.

Inicialmente el protocolo y SDK de Z-Wave no estaban abiertos y solamente estaban disponibles para aquellos fabricantes de dispositivos que hubiesen firmado un acuerdo de confidencialidad con Sigma Designs. Este acuerdo evitaba que los fabricantes hiciesen público el contenido del SDK. En 2015 se publica el standard internacional ITU-T G.9959 de las capas física y MAC del protocolo Z-Wave y ya en 2016 Sigma Designs decide abrir la capa de interoperabilidad de la especificación Z-Wave. Es a partir de este hito cuando se empieza a crear una librería con licencia LGPL llamada Open Z-Wave.

Z-Wave opera en la banda de radiofrecuencia ISM (industrial, científica y médica), concretamente en frecuencias de 868,42 MHz para Europa y 908,42 MHz en Estados Unidos. Estas frecuencias están diseñadas para comunicaciones de datos de ancho de banda bajo en dispositivos integrados como sensores de seguridad, alarmas y paneles de automatización. Al igual que ZigBee, cada componente Z-Wave es un repetidor, en este caso de RF, construyendo una red de malla de dispositivos inalámbricos llegando incluso a una distancia máxima de 400 metros.

Dentro de esta malla se pueden encontrar dos tipos de dispositivos Z-Wave, los controladores y los esclavos. Los dispositivos controladores son los que inician los comandos de control enviándolos a otros nodos y los dispositivos esclavos son los que finalmente ejecutan las acciones. Estos dispositivos esclavos también tiene la capacidad de reenviar los mensajes a otros dispositivos esclavos, de este modo el dispositivo controlador puede comunicarse con otros nodos que no tenga visibilidad directa. Este enrutamiento se realiza de manera automática.

Puede darse el caso que un nodo no esté disponible cuando un controlador inicie la comunicación, en ese caso, el nodo caído se añade a una lista “failed-node-list”. En ese punto,

el usuario puede decidir qué hacer, eliminar el nodo incomunicado o bien solicitar al controlador que vuelva a escanear la red actualizando la lista de los nodos vecinos.

### 2.4.3. ESP-NOW

Además de las tecnologías enumeradas anteriormente, existe un protocolo propietario de la empresa Espressif Systems que trabaja en la frecuencia de 2,4 Ghz, la misma frecuencia que el Wi-Fi y Bluetooth. La empresa implementó este protocolo para poder emparejar sus microcontroladores ESP32 y ESP8266 antes de realizar la comunicación de datos. De este modo un mismo dispositivo puede enviar y recibir datos desde y hacia varios microcontroladores (figuras 9, 10 y 11). Este protocolo tiene alguna limitación en cuanto al número de dispositivos emparejados, si la comunicación no está cifrada puede llegar a emparejar hasta 20 microcontroladores y si la comunicación está cifrada el número de dispositivos emparejados se reduce a 10.

En cuanto a la conectividad, el protocolo ESP-NOW permite al microcontrolador volver a emparejarse de manera automática a la red ante una pérdida de alimentación. Según una práctica de campo realizado por Cameron (2021) consiguió llegar a un alcance de transmisión de 250 metros sobre terreno abierto con solamente dos microcontroladores ESP32.

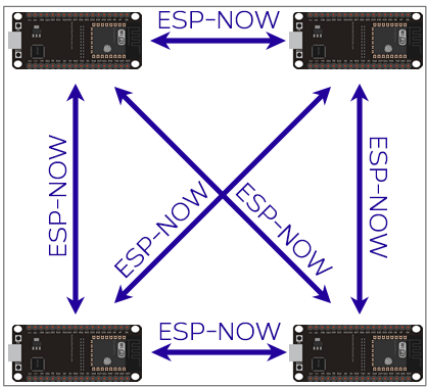
***Figura 9.*** *Receptor de mensajes con ESP-NOW.*


Fuente: Elaboración propia, 2021.

***Figura 10.*** *Emisor de mensajes con ESP-NOW.*


Fuente: Elaboración propia, 2021.

***Figura 11.*** *Emisor y receptor de mensajes con ESP-NOW.*




Fuente: Elaboración propia, 2021.

## 2.5. Hardware

### 2.5.1. ESP8266

El ESP8266 es un microcontrolador SoC de bajo costo con Wi-Fi desarrollado por la empresa Espressif Systems. Fue en 2014 cuando la empresa AI-Thinker saca al mercado la primera versión ESP-01 y desde entonces hay un listado enorme de versiones del ESP8266 (ESP-01, ESP-02, ESP-03,…, ESP-12). Las versiones más comunes que se pueden encontrar son la ESP-01 y la ESP-12.

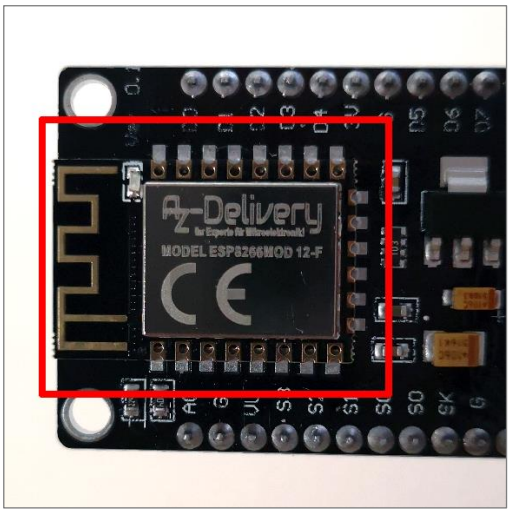
Este microcontrolador lleva un procesador integrado Tensilica LX106 de 32 bits con arquitectura RISC a 80 MHz y que puede alcanzar una velocidad máxima de 160 MHz. En cuanto a la memoria flash, esto dependerá del módulo sobre el que se montará este SoC. Cabe destacar que la memoria podría oscilar entre módulos partiendo de 1 MB hasta 8 MB. También existen módulos que pueden llegar hasta los 16MB.

**Tabla 3. Características principales del ESP8266-12E.**

| Chip | ESP8266-12E |
|---|---|
| CPU | Tensilica LX106 32 bit at 80 MHz (up to 160 MHz) |
| SRAM | 36 KB available |
| FLASH | 4MB (max. 16MB) |
| Voltage | 3.0V to 3.6V |
| Operating Current | 80 mA average |
| Programmable | Free (C, C++, Lua, etc.) |
| Open source | Yes |
| Wi-Fi | 802.11 b/g/n |
| Bluetooth® | - |
| UART | 2 |
| GPIO | 17 |
| SPI | 2 |
| I2C | 1 |
| PWM | - |
| ADC | 1 (10-bit) |
| DAC | - |
| Size | 24.0 x 16.0 x 3.0 mm |
| Prize | £5 |

Fuente: Maier, Sharp, & Vagapov, 2017.

***Figura 12.*** *ESP8266 versión 12-F montado sobre placa NodeMCU V3.*




Fuente: Elaboración propia, 2021.

### 2.5.2. ESP32

Podemos considerar el ESP32 como el hermano mayor del anteriormente citado ESP8266. Este SoC está desarrollado por la misma empresa. Se diferencia de su antecesor en que dispone de mayor memoria, mayor potencia (usa un procesador de doble núcleo), más puertos de entrada y salida y añade la tecnología Bluetooth®.

**Tabla 4. Características principales del ESP-WROOM-32.**

| Chip | ESP-WROOM-32E |
|---|---|
| CPU | Tensilica Xtensa LX6 32 bit Dual-Core at 160/240 MHz |
| SRAM | 520 KB |
| FLASH | 2MB (max. 64MB) |
| Voltage | 2.2V to 3.6V |
| Operating Current | 80 mA average |
| Programmable | Free (C, C++, Lua, etc.) |
| Open source | Yes |
| Wi-Fi | 802.11 b/g/n |
| Bluetooth® | 4.2 BR/EDR + BLE |
| UART | 3 |
| GPIO | 32 |
| SPI | 4 |
| I2C | 2 |
| PWM | 8 |
| ADC | 18 (12-bit) |
| DAC | 2 (8-bit) |
| Size | 25.5 x 18.0 x 2.8 mm |
| Prize | £8 |

Fuente: Maier, Sharp, & Vagapov, 2017.

El salto en cuanto a potencia y número y tipo de puertos de entrada/salida respecto al ESP8266 se postula como un gran candidato a la hora de realizar proyectos y aplicaciones IoT. Por ello muchos fabricantes están desarrollando varias placas de desarrollo integrando este SoC. Podemos encontrar placas con batería incluida, con pantallas OLED, pantallas TFT, etc.

Este microcontrolador tiene integrados sensor de temperatura, sensor de efecto Hall y un controlador de mando a distancia por infrarrojos de 8 canales. Otro aspecto interesante son las salidas PWM para control de LED (16 salidas) y motores (1 salida) permitiendo, en este último caso, regular la velocidad de giro del propio motor.

***Figura 13.*** *ESP32 versión WROOM-32 montado sobre placa de desarrollo.*


Fuente: Elaboración propia, 2021.

### 2.5.3. NodeMCU

Los SoC ESP32 y ESP8266, salvo que se utilicen en cadenas de montaje dónde por costes y espacio es mejor utilizar el propio microcontrolador, suelen estar montados sobre placas de desarrollo. Esto permite cargar los programas de manera sencilla (normalmente a través de USB), alimentación a través del propio USB y terminales (pines) para conectar sensores y actuadores de manera más sencilla. También dispone de un botón de reset y un led integrado que puedes personalizar en tus programas.

La placa de desarrollo más común que podemos encontrar cuando buscamos por internet es la NodeMCU. Originalmente este nombre se refería al propio firmware Open Source. Desde 2015 el equipo de desarrollo dejó de mantener el firmware y es la comunidad de desarrolladores quienes la mantienen, debido a este motivo, cuando se habla de NodeMCU a día de hoy nos referimos más a la placa de desarrollo que al firmware. Existen tres versiones de NodeMCU:

- Primera generación V0.9 (versión obsoleta).
- Segunda generación V1.0 / V2 (versión oficial de NodeMCU).
- Tercera generación V1.0 / V3 (versión mejorada y la más vendida).

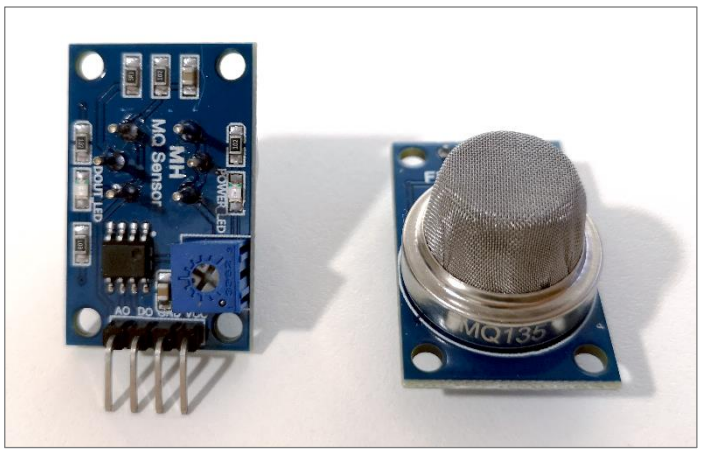
### 2.5.4. MQ-135

Dentro del grupo de sensores que podemos conectar a los microcontroladores, la serie MQ se corresponde al grupo de sensores de detección de diferentes tipologías de gases. De manera más específica, el sensor MQ-135 está especialmente indicado para medir los niveles de gases peligrosos como son los del tipo NOx, amoníaco (NH3), benceno, alcohol, humo y niveles de CO.

Estos sensores requieren de un precalentamiento inicial para eliminar cualquier resto de humedad o restos que puedan quedar en su interior. Para ello el fabricante recomienda dejar encendido el sensor durante sus primeras 24h de manera ininterrumpida. Este proceso solamente es necesario realizarlo la primera vez que se utilice. Posteriormente, estos sensores empiezan a dar lecturas fiables después de estar 5 minutos en marcha. Este precalentamiento es necesario debido a que el módulo dispone de una cámara de calentamiento por donde debe entrar y salir el gas. Estos sensores son electroquímicos y varían su resistencia cuando un gas pasa por los sensores variando su resistencia.

En cuanto a la conexión, el sensor dispone de cuatro pines, salida analógica, salida digital, tierra y tensión (entre 2.5 y 5V), un potenciómetro para calibración, un led de encendido y un led cuando la salida digital está activa por detección del gas (figura 14).

***Figura 14.*** *Sensores de gas MQ-135.*




Fuente: Elaboración propia, 2021.

## 2.6. Software

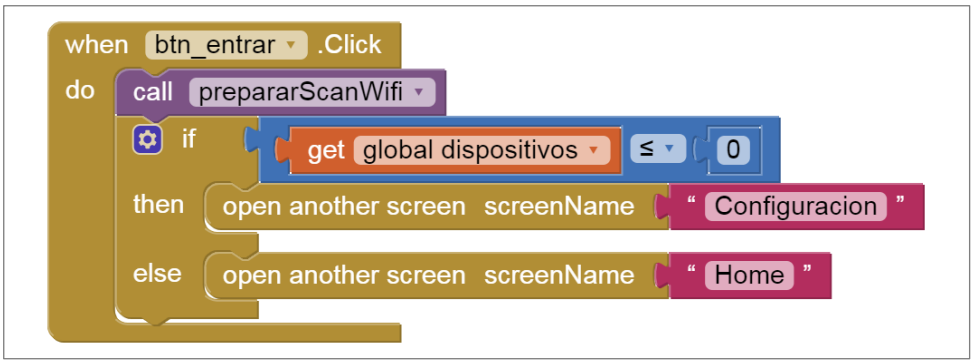
### 2.6.1. App Inventor

App Inventor es una aplicación de programación basada en bloques, originalmente desarrollada por Google y posteriormente mantenida por el Massachusetts Institute of Technology (MIT). Fue creada en 2009 y dirigida por Harold Abelson y Mark Friedman, profesores del MIT. En 2011 Google anuncia que deja de dar soporte a la aplicación y pasa a ser Open Source.

Las aplicaciones creadas con este programa, debido a su simplicidad, tiene algunas limitaciones, no obstante, permite realizar aplicaciones para dispositivos móviles Android de manera muy sencilla y realmente útiles. La naturaleza visual, basada en eventos, reduce los primeros problemas de sintaxis comunes cuando un principiante se enfrenta al mundo de la programación, pudiendo crear aplicaciones casi de manera inmediata y con gran utilidad del mundo real (Wolber, 2011).

El ámbito de aplicación de App Inventor abarca desde el mundo de la educación creando aplicaciones educativas, pasando por la creación de prototipos de aplicaciones para plasmar ideas o construir una aplicación completa que resuelve un problema del mundo real (Wolber, Hal, Spertus, & Looney, 2011).

***Figura 15.*** *Ejemplo de código manejando el evento de click sobre un botón.*




Fuente: App Inventor, 2021.

En cuanto a las posibilidades de diseño, App Inventor dispone de los elementos básicos en cuanto a interfaz de usuario como pueden ser botones, etiquetas, notificaciones, listas, webviews, switch, campos de texto, imágenes, selector de fecha,… Además, dispone de opciones para acceder a los distintos sensores del móvil como son el acelerómetro, termómetro, sensor de proximidad, NFC, sensor de humedad, giroscopio, etc.

También, en referencia al almacenamiento de datos persistentes, permite conectarte con una pequeña base de datos creada cuando se ejecuta la aplicación en el terminal, o bien una base de datos en la web, uso de ficheros o una base de datos en la nube. Asimismo permite usar el Bluetooth®, el puerto serie para conectar con dispositivos Arduino y uso de servicios API REST.

Si además se necesita de funcionalidades o bloques no disponibles en el propio programa, existe la opción de añadir extensiones. Al tratarse de una aplicación de código abierto, basada en Open Blocks de Java, permite implementar bloques personalizados. Uno de los repositorios de extensiones más utilizados por los usuarios de App Inventor es la web Pura Vida Apps[2] en el que podrás encontrar cualquier tipo de extensión como por ejemplo manejo de redes Wi-Fi, notificaciones push, uso de SMS, etc.

### 2.6.2. Arduino CC

Cuando hablamos de Arduino lo primero que nos viene a la cabeza es la placa con el microcontrolador programable y sus pines de conexión de sensores. Sin embargo, como bien señala en su libro Artero (2013), Arduino abarca tres conceptos:

- Una placa hardware libre: Se trata de una placa hardware con un microcontrolador reprogramable con pines hembra (analógicos y digitales) para conexión de distintos sensores y actuadores.

[2] URL del sitio: https://puravidaapps.com

- Un software gratis, libre y multiplataforma: Entorno de desarrollo que nos permite elaborar distintos programas y cargarlos posteriormente a nuestra placa de desarrollo mediante el puerto USB.

- Un lenguaje de programación libre: Dentro de este lenguaje de programación libre podemos encontrar elementos parecidos a otros lenguajes de programación como bloques condicionales, variables, bloques iterativos, etc. Aunque Arduino está basado en Processing, que usa internamente código Java, el lenguaje de programación es C/C++.

Centrándonos en el propio software, el entorno de desarrollo o IDE, Arduino CC es el software que permite elaborar programas y posteriormente cargarlos a las placas de desarrollo. Este programa dispone de un gestor de placas que permite la compatibilidad con distintas placas, como por ejemplo la ESP32 o la ESP8266. Asimismo, tiene de un gestor de librerías que facilita el desarrollo de programas incluyendo extensiones como por ejemplo la librería Arduino Json que permite trabajar en formato JSON[3] dentro del propio programa (crear objetos, variables, serializar,…).

## 2.7. Resumen y conclusiones del estado del arte

Tras un análisis de la problemática actual referente a las muertes por inhalación involuntaria de gases tóxicos, se puede proponer una solución IoT que ayude a prevenir estos accidentes. Actualmente existen muchas soluciones de control de CO en el hogar, con multitud de aplicaciones para dispositivos móviles, pero todas ellas son cerradas y propietarias y normalmente requieren de acceso a internet para hacer uso de ellas.

Para elaborar una solución barata, libre y accesible a todas las personas, utilizando los microcontroladores ESP32 y ESP8266 y el uso de sensores de detección de gas como el MQ-

[3] JavaScript Object Notation

135, se puede plantear una propuesta de arquitectura que permita monitorizar y mitigar problema planteado. Todo ello desarrollado mediante el IDE de Arduino y App Inventor.

# 3. Objetivos y Metodología de Trabajo

## 3.1. Objetivo General

El objetivo de este trabajo final de máster es elaborar un prototipo de aplicación para dispositivos móviles Android capaz de monitorizar los niveles de calidad del aire detectados por un sensor. En base a esos datos la aplicación tendrá la autonomía suficiente para avisar al usuario mediante una notificación push a su teléfono móvil cuando detecte que el sensor supera un umbral concreto.

Esta aplicación, en contrapunto al resto de aplicaciones comerciales existentes, se diferencia en que no requiere de conectividad a internet para que se pueda utilizar. El propósito de esta aplicación es convertir el hogar en un lugar más seguro, mejorando la accesibilidad del mismo para aquellas personas en situación de dependencia o que correspondan a un colectivo vulnerable (tecnológicamente hablando) como pueden ser las personas mayores.

Además de tratarse de una aplicación open source, apoyándonos con los microcontroladores ESP32 y el ESP8266, puede convertirse en una solución de bajo costo económico y energético. Estos dispositivos apenas consumen energía y su coste económico es relativamente bajo.

Para la consecución del objetivo principal, el trabajo se ha dividido en 5 objetivos más específicos.

## 3.2. Objetivos Específicos

### 3.2.1. Creación de la APP

Con este objetivo específico se pretende desarrollar una aplicación usando el entorno de desarrollo App Inventor. Esta aplicación será capaz de interpretar la información recibida del servidor. También dispondrá de la funcionalidad de enviar notificaciones push al dispositivo cuando ésta detecte niveles altos de CO.

### 3.2.2. Creación de un servidor web

Para que el primer objetivo específico se pueda conseguir, es necesario implementar un servidor web utilizando el microcontrolador ESP32. Este pequeño servidor podrá manejar las peticiones recibidas y enviar la respuesta con los datos solicitados.

### 3.2.3. Comunicación entre ESP32 y ESP8266

En este objetivo específico se intenta desarrollar la comunicación entre los microcontroladores ESP32 y los ESP8266. Este modo de comunicación entre ambos SoC se realizará a través del bus serie que disponen estos dispositivos. Para este intercambio de información, se pretende desarrollar un protocolo de comunicación ad-hoc para interpretar los mensajes enviados por el ESP8266 al ESP32.

### 3.2.4. Comunicación entre microcontroladores ESP8266

Utilizando el protocolo propietario ESP-NOW, este objetivo específico intenta establecer una comunicación inalámbrica entre distintos ESP8266. Por otra parte, también tiene el objetivo de almacenar la información recopilada durante la comunicación entre los dispositivos.

### 3.2.5. Comunicación con el sensor MQ-135

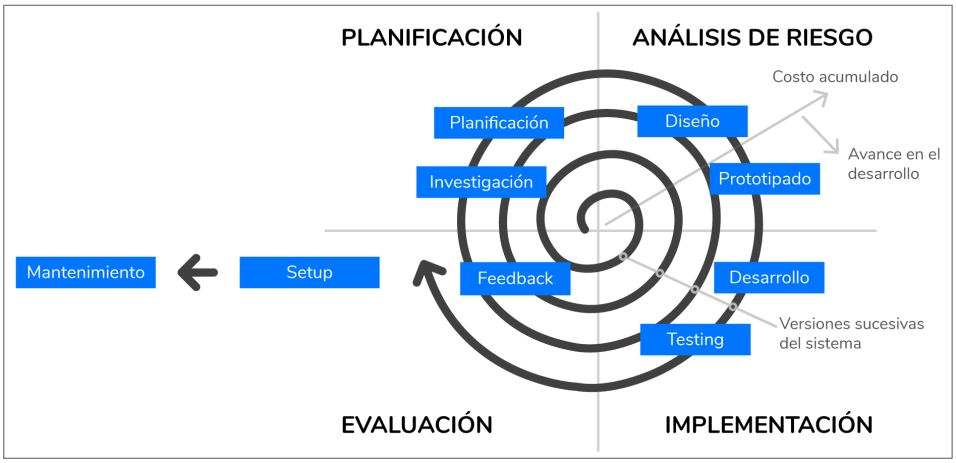
Este objetivo específico es el primero en la cadena para la consecución del objetivo principal. Para llevar a cabo este propósito, se pretende comunicar el sensor MQ-135 a través

del pin de entrada analógica del ESP8266. Tras esta comunicación, tratar y almacenar los datos recibidos en la propia memoria del SoC.

## 3.3. Metodología de Trabajo

La metodología que se empleará en este proyecto es la metodología en espiral. Esta metodología de trabajo es una combinación entre el modelo en cascada y el modelo iterativo o basado en prototipos. Cada iteración de la espiral se corresponde con el cumplimiento de un objetivo específico, de esta manera empiezas con objetivos de baja complejidad y se va incrementando con forme se va avanzando el proyecto y se van cumpliendo el resto de objetivos (figura 16).

***Figura 16.*** *Diagrama de metodología en espiral.*




Fuente: ASPgems, 2021.

En cuanto a las fases, la primera fase o fase de planificación debemos investigar sobre el problema a resolver y se establece una planificación si realmente es necesario. La segunda fase, el análisis de riesgo, se evalúan todos aquellos puntos dónde pueda suponer un riesgo para el desarrollo del proyecto. También, si se requiere, se realiza un pequeño prototipo. En la fase de implementación es la fase dónde se empieza con la codificación y las pruebas. Por último, en la fase de evaluación se realiza un análisis sobre el alcance de los objetivos planteados al inicio del ciclo y si se han cumplido. En caso de detectar problemas, deberá tenerse en cuenta en futuras iteraciones.

Si intentamos encajar la secuencia de objetivos específicos de nuestro proyecto en una metodología en espiral, los ciclos quedarían de la siguiente manera:

- Ciclo 1: Comunicación con el sensor MQ-135.
- Ciclo 2: Comunicación entre microcontroladores ESP8266.
- Ciclo 3: Comunicación entre ESP32 y ESP8266.
- Ciclo 4: Creación de un servidor web.
- Ciclo 5: Creación de la APP.

# 4. Desarrollo Específico de la Contribución

## 4.1. Planificación

Del mismo modo que en cualquier proyecto software, para la realización de este trabajo fin de máster también se ha elaborado una planificación de las distintas etapas de las que se constituye este proyecto. Con el diagrama de Gantt se han distribuido todas las tareas y se han planificado los tiempos de cada una. Cada bloque representa una semana de trabajo considerando el inicio del proyecto el 1 de marzo de 2021 (figura 17).

***Figura 17.*** *Diagrama de Gantt sobre la planificación del proyecto.*

| Tarea | Descripción | S 01 | S 02 | S 03 | S 04 | S 05 | S 06 | S 07 | S 08 | S 09 | S 10 | S 11 | S 12 | S 13 | S 14 | S 15 | S 16 | S 17 | S 18 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| INVESTIGACIÓN | | | | | | | | | | | | | | | | | | | |
| 1.1 | Investigación del problema: calidad del aire | | | | | | | | | | | | | | | | | | |
| 1.2 | Investigación sobre las soluciones actuales | | | | | | | | | | | | | | | | | | |
| DOCUMENTACIÓN TÉCNICA | | | | | | | | | | | | | | | | | | | |
| 2.1 | Estudio sobre ESP32 y ESP8266 | | | | | | | | | | | | | | | | | | |
| 2.2 | Estudio sobre sensor MQ-135 | | | | | | | | | | | | | | | | | | |
| 2.3 | Estudio sobre App Inventor | | | | | | | | | | | | | | | | | | |
| ANÁLISIS | | | | | | | | | | | | | | | | | | | |
| 3.1 | Definición de los requisitos funcionales | | | | | | | | | | | | | | | | | | |
| 3.2 | Definición de los requisitos no funcionales | | | | | | | | | | | | | | | | | | |
| 3.3 | Elaboración del prototipo de la APP | | | | | | | | | | | | | | | | | | |
| 3.4 | Elaboración de los casos de uso | | | | | | | | | | | | | | | | | | |
| DESARROLLO | | | | | | | | | | | | | | | | | | | |
| 4.1 | Elaboración de los diagramas de secuencia | | | | | | | | | | | | | | | | | | |
| 4.2 | Elaboración del interfaz de usuario | | | | | | | | | | | | | | | | | | |
| 4.3 | Diseño y desarrollo de la lógica de los SoCs | | | | | | | | | | | | | | | | | | |
| 4.4 | Diseño y desarrollo de la lógica de la APP | | | | | | | | | | | | | | | | | | |
| PRUEBAS | | | | | | | | | | | | | | | | | | | |
| 5.1 | Evaluación de los requisitos funcionales | | | | | | | | | | | | | | | | | | |
| 5.2 | Evaluación de los requisitos no funcionales | | | | | | | | | | | | | | | | | | |
| CONCLUSIONES | | | | | | | | | | | | | | | | | | | |
| 6.1 | Evaluación de conclusiones | | | | | | | | | | | | | | | | | | |
| 6.2 | Mejoras y futuras líneas de investigación | | | | | | | | | | | | | | | | | | |

Fuente: Elaboración propia, 2021.

## 4.2. Identificación de requisitos y casos de uso

Uno de los pasos necesarios en cuanto al diseño de una solución es determinar cuáles son los requisitos que debe cumplir. Estos requisitos se pueden clasificar en dos grupos, los requisitos funcionales y los no funcionales. Dentro del primer grupo se sitúan los requisitos específicos de interacción del sistema con el usuario, es decir, cómo debe funcionar la aplicación y qué se espera de ella. Sobre el segundo grupo, los requisitos no funcionales, son requisitos transparentes al usuario final, como por ejemplo, eficiencia del código, manejo de errores, protocolos de comunicación entre componentes, etc.

### 4.2.1. Requisitos funcionales

#### 4.2.1.1. Gestión de dispositivos

- RF-1: El usuario podrá añadir los dispositivos nuevos identificándolos con un nombre. No podrá haber una misma MAC asociada a dos nombres distintos.
- RF-2: El usuario podrá modificar el nombre del dispositivo guardado anteriormente.
- RF-3: El usuario podrá borrar los dispositivos existentes.

#### 4.2.1.2. Visualización del estado de calidad del aire del sensor

- RF-4: El usuario podrá seleccionar el dispositivo que quiere visualizar desde una lista desplegable.
- RF-5: La aplicación mostrará si la calidad del aire es “buena” o “mala” con mediante un literal. La decisión de mostrar un literal u otro dependerá del umbral definido por el usuario.
- RF-6: La aplicación mostrará con un rosco de progresión la calidad del aire cambiando de color si supera un umbral.

#### 4.2.1.3. Visualización de las alertas

- RF-7: El usuario podrá configurar el umbral sobre el que recibirá las alertas.
- RF-8: La aplicación deberá mostrar las alertas cuando el valor de alguno de los sensores supere el umbral que ha definido el usuario.

### 4.2.2. Requisitos no funcionales

#### 4.2.2.1. Gestión automática del listado de dispositivos

- RNF-1: El listado de dispositivos de la pantalla de configuración se actualizará automáticamente cuando el usuario añada, modifique o borre un dispositivo.
- RNF-2: El listado de dispositivos de la pantalla de visualización se actualizará automáticamente cuando el usuario añada, modifique o borre un dispositivo.
- RNF-3: La aplicación, además de guardar el listado de sensores en la propia aplicación, los guardará también en los servidores web ESP32 para que vayan almacenando la información recopilada por los sensores.

#### 4.2.2.2. Refresco automática de la lectura de sensores

- RNF-4 La aplicación deberá refrescar los datos de lectura de los sensores de manera automática.

#### 4.2.2.3. Conexión automática al punto de acceso

- RNF-5: La aplicación, una vez iniciada, se conectará al punto de acceso con mayor señal. En caso de no poder conectar con ningún punto de acceso no mostrará ningún dato.

#### 4.2.2.4. Control de los datos recibidos en el servidor por los sensores

- RNF-6: Los distintos servidores ESP32 que reciben los datos de los sensores deberán controlar que la MAC está dentro del listado que previamente la APP habrá establecido. En caso que no exista, no guardará el valor.
- RNF-7: Los distintos servidores ESP32 guardarán, además del valor del sensor, el tiempo sobre el que ha recibido la lectura.

#### 4.2.2.5. Protocolo de comunicación del puerto serie

- RNF-8: El servidor web ESP32 deberá interpretar los datos recibidos por el puerto serie, descartando aquellos datos que sean incongruentes o incompletos.
- RNF-9: Los SoC ESP8266 deberán enviar por el puerto serie la información recibida por el propio sensor y también la información recibida de otros ESP8266.

#### 4.2.2.6. Conexión automática con ESP-NOW

- RNF-10: Los distintos dispositivos ESP8266, que son los encargados de recibir la información directa del sensor, deberán enviar y recibir la información de manera automática cada vez que arranquen.

#### 4.2.2.7. Eficiencia del código fuente

- RNF-11: El código fuente de la aplicación, desarrollado en App Inventor, deberá ser eficiente, usando procedimientos para simplificar y evitar la repetición de código, uso eficiente de la memoria en base de datos, etc.
- RNF-12: El código fuente de los microcontroladores, desarrollado con Arduino, deberá ser eficiente, usando librerías existentes, creando funciones evitando duplicidad de código, etc.

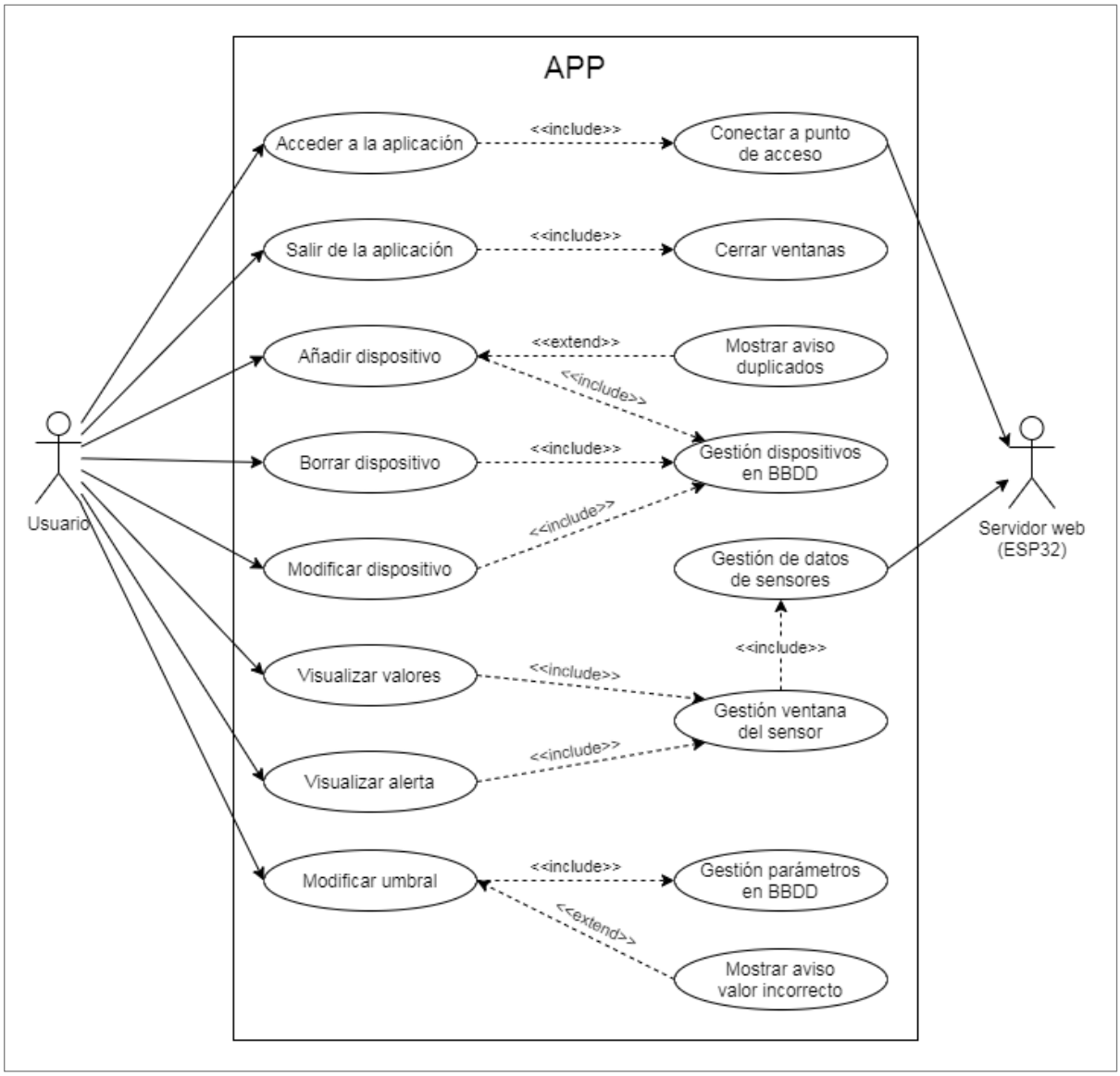
### 4.2.3. Casos de uso

A continuación se detallarán los distintos casos de uso de la aplicación. Estos casos son interacciones realizadas por un actor (ya sea el propio usuario como la APP) en el sistema. La idea de la elaboración de este diagrama es definir todas aquellas acciones que se puedan dar en el sistema facilitando la comprensión del mismo. Para la representación gráfica de estos casos de uso se ha utilizado el Lenguaje Unificado de Modelado (UML).

#### 4.2.3.1. Caso de uso de la APP

En la figura 18 se representa el esquema de los casos de uso que se pueden recoger dentro del sistema de la APP. Como se puede comprobar existen dos actores, el primer actor representa el usuario que utiliza la app y el segundo actor representa el servidor web. El primer actor puede acceder a la APP, gestionar los dispositivos, visualizar los valores y visualizar y gestionar las alertas. Respecto al segundo actor es el encargado de recibir las peticiones de conexión como punto de acceso y recibir las peticiones de gestión y obtención de valores de los sensores.

***Figura 18.*** *Caso de uso de la APP.*



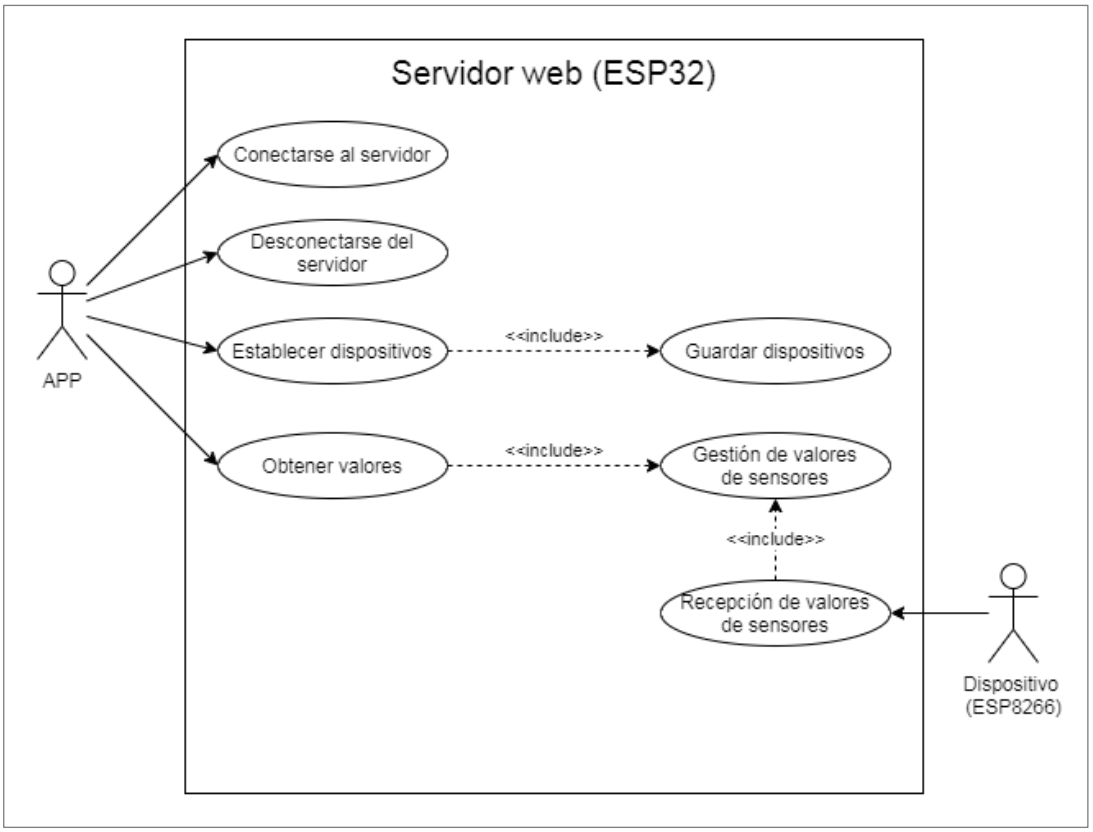

Fuente: Elaboración propia, 2021.

En cuanto a las actividades o procesos secundarios más destacables, se puede observar la gestión de dispositivos en BBDD que representa la gestión en la base de datos interna de la aplicación dónde se almacenará todos los datos de los sensores administrados. También existe otra actividad relacionada con la gestión de los parámetros de BBDD que será la encargada de gestionar los parámetros básicos de la aplicación como pueden ser el número de dispositivos, nombres, umbral, etc.

#### 4.2.3.2. Caso de uso del servidor web

En la figura 19 se puede observar el esquema de los casos de uso que recoge el entorno del servidor web (ESP32). En este caso intervienen dos actores, por una parte la aplicación cuando se conecta y desconecta al servidor, establece los sensores del usuario y solicita los valores de los sensores. Por otra parte está el actor que representa el dispositivo que recibe el valor del sensor propio y si aplica, el del resto de dispositivos conectados a la red.

***Figura 19.*** *Caso de uso del servidor web (ESP32).*



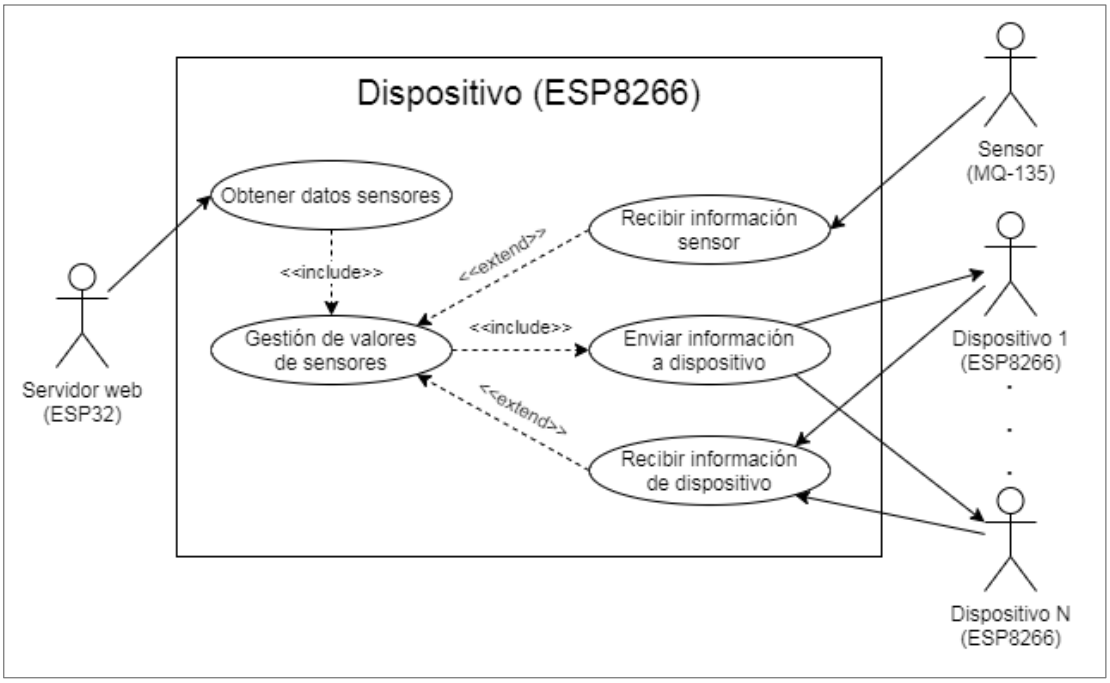

Fuente: Elaboración propia, 2021.

Respecto a las actividades secundarias encontramos el guardado de dispositivos encargada de almacenar los dispositivos que ha introducido el usuario en la aplicación, de este modo cuando el servidor reciba los valores de los dispositivos se compruebe si está en esa lista, si coincide la MAC guarda el valor, en caso contrario lo rechaza. En cuanto a la segunda actividad secundaria es la encargada de la gestión de los valores de los sensores, es decir, es la responsable de interpretar los datos recibidos por el dispositivo conectado físicamente al servidor y almacenarlo comprobando si el valor corresponde a alguno de los dispositivos almacenados.

#### 4.2.3.3. Caso de uso del dispositivo

En la figura 20 se representa el sistema correspondiente al propio dispositivo, será el encargado de recibir la información del sensor y de otros dispositivos conectados a la red. En este caso de uso se han contemplado varios actores ya que puede variar en función de los dispositivos conectados a él.

***Figura 20.*** *Caso de uso del dispositivo (ESP8266).*



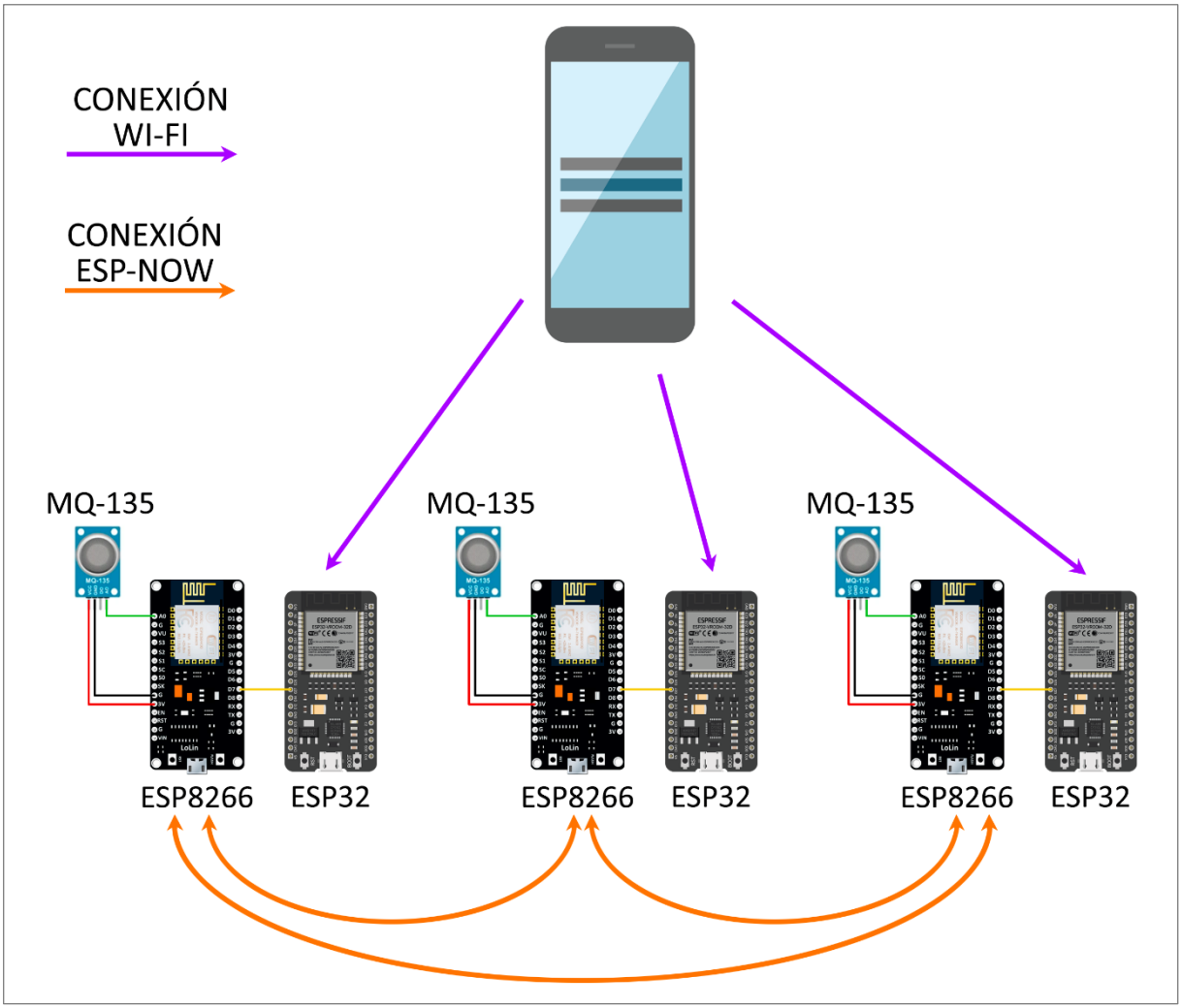

Fuente: Elaboración propia, 2021.

Por una parte, el primer actor es el servidor web, responsable de solicitar los datos al dispositivo de los que dispone en ese momento. Por otra parte se encuentra el sensor de gas, éste es el encargado de enviar la información recogida en todo momento. Por último están los actores, que puede ser ninguno, uno o muchos. Éstos últimos representan a los otros dispositivos conectados a la red creada y envian la información de su sensor y recibir la información de otros sensores conectados a otros dispositivos.

## 4.3. Descripción de la arquitectura y prototipo

### 4.3.1. Arquitectura física

***Figura 21.*** *Diseño de la arquitectura física de la solución.*




Fuente: Elaboración propia, 2021.

En la figura 21 podemos encontrar el diseño de la arquitectura física de este trabajo fin de máster. En ella podemos encontrar el propio dispositivo móvil que tiene instalada la app, los sensores de gas, los ESP8266 y los ESP32. Aunque con un único sensor, un único ESP32 y un único ESP8266 sería suficiente para realizar la lectura de un sensor y mostrarlo por la aplicación, se ha decidido añadir tres dispositivos. El motivo ha sido poder ir conectando y desconectando los distintos puntos de acceso y así comprobar que la conexión a través de ESP-NOW se realiza de manera automática y funciona correctamente.

Los dispositivos ESP32 serán los encargados de arrancar el servidor web, recibir las peticiones HTTP a través de la conexión inalámbrica con el móvil y recibir los datos de los sensores a través de su puerto serie (marcado en la figura con una conexión amarilla). Los dispositivos ESP8266 serán los encargados de recibir la información del sensor MQ-135 (marcado en la figura con la conexión verde a la entrada analógica) y recibir el valor de otros sensores enviados por otros dispositivos a través del protocolo inalámbrico ESP-NOW.

### 4.3.2. Estructura BBDD

Para poder almacenar los datos recogidos de los sensores y algunas configuraciones de la propia aplicación, App Inventor ofrece dos alternativas: almacenamiento en fichero o almacenamiento en base de datos. Por la tipología de información a manejar se ha decidido utilizar el almacenamiento en base de datos. En este caso App Inventor ofrece varias posibilidades, utilizar una base de datos en el propio dispositivo o utilizar una base de datos de internet o en la nube. Como la idea de la aplicación era que fuese funcional sin el uso de internet se ha optado por utilizar el componente TinyDB de App Inventor que te permite montar una base de datos no relacional en el propio dispositivo pudiendo guardar datos de manera persistente.

De cara al diseño de la estructura de datos se ha tenido en cuenta la tipología de base de datos que permite App Inventor. Esta base de datos está organizada en namespaces y cada namespace se estructura como clave-valor. Tanto la clave como el valor pueden ser numérico o alfanumérico sin necesidad de especificar la tipología previamente.

En cuanto al número de namespaces o tablas se han identificado tres tipos. El primer namespace identificado es para almacenar datos de configuración o globales de la aplicación, como por ejemplo el número de dispositivos, el umbral de calidad del aire, la contraseña del punto de acceso, etc.

El otro namespace identificado será el encargado de almacenar la dirección MAC introducida por el usuario con el nombre que ha establecido en la pantalla de configuración. Por último, se crearán tantos namespaces como dispositivos haya creados, siendo el nombre del namespace la dirección MAC del dispositivo. En dicho namespace se almacenará la fecha en la que se ha recibido el valor y el valor obtenido.

***Figura 22.*** *Estructura de los namespaces de la aplicación.*

| namespace: <<dirección MAC>> | | |
|---|---|---|
| Clave | Valor | Descripción |
| <<fecha>> | numérico | Guarda el valor del sensor en una fecha concreta |

| namespace: datosDispositivos | | |
|---|---|---|
| Clave | Valor | Descripción |
| <<dirección mac>> | alfanumérico | Guardar los nombres asociados a las direcciones MAC |

| namespace: datosGeneral | | |
|---|---|---|
| Clave | Valor | Descripción |
| numDispositivos | numérico | Guardar la cantidad total de números de dispositivos guardados |
| passwordAP | alfanumérico | Guardar la contraseña del punto de acceso |
| umbral | numérico | Guardar el umbral que no debe rebasar la calidad del aire |
| haSidoModificado | booleano | Flag que indica si se ha modificado la lista de dispositivos |

Fuente: Elaboración propia, 2021.

Los campos marcados como << >> son campos dónde el valor será definido en tiempo de ejecución de la aplicación. Por ejemplo, el nombre del namespace que almacena el valor y la fecha está identificado como *<<dirección MAC>>* y en tiempo de ejecución de la aplicación

dependerá del valor de la dirección MAC que haya introducido el usuario como dispositivo asociado (por ejemplo 11:22:33:44:55:66). En este mismo namespace existe la clave *<<fecha>>* donde el nombre de la clave tendrá el valor del tiempo en milisegundos en el momento de obtención del valor.

### 4.3.3. Prototipo

Otro paso importante antes de empezar con los desarrollos es definir los prototipos de las pantallas que formarán parte de la aplicación así como el comportamiento de cada componente. En esta aplicación existirán tres pantallas (figura 23, 24 y 25) que se corresponde a la pantalla inicial, la pantalla de configuración de dispositivos y la pantalla de visualización de datos de los sensores.

#### 4.3.3.1. Pantalla de inicio

***Figura 23.*** *Prototipo de la pantalla de inicio.*


Fuente: Elaboración propia, 2021.

La pantalla inicial (figura 23) está compuesta por un mensaje de bienvenida a la aplicación y dos botones, acceso a la aplicación o salir. El botón salir cerrará la aplicación y cualquier ventana que haya abierta. Por otra parte, el botón de entrar, antes de mostrar cualquier dato, realizará una conexión con el punto de acceso con mayor señal, si no conecta mostrará una alerta que impedirá al usuario acceder al resto de aplicación. En caso que pueda conectar con el punto de acceso, si el usuario no ha definido ningún dispositivo le mostrará la pantalla de configuración (figura 24), por el contrario, le mostrará la pantalla de visualización de datos (figura 25).

#### 4.3.3.2. Pantalla de configuración de dispositivos

***Figura 24.*** *Prototipo de la pantalla de configuración de dispositivos.*


Fuente: Elaboración propia, 2021.

En la pantalla de configuración de dispositivos (figura 24) el usuario podrá visualizar la información, borrar y añadir nuevos dispositivos. En cuanto a la visualización de información de los distintos dispositivos, el usuario podrá seleccionar del desplegable el nombre del

dispositivo que desea seleccionar. Una vez seleccionado se habilitan los botones de “Más Info” y “Borrar”. El primer botón visualiza la dirección MAC asociada al dispositivo y el segundo borra el dispositivo de la lista de dispositivos guardados.

La sección de nuevo dispositivo, el usuario puede agregar el dispositivo del que quiere obtener información. En esta sección el usuario deberá introducir la dirección MAC y el nombre asociado. El botón de “Añadir” guarda en la base de datos el nombre y dirección MAC siempre y cuando los dos campos estén completos y no exista ninguna dirección MAC almacenada que coincida con la que se quiera introducir. En caso que no se cumpla alguna de las condiciones anteriores, se mostrará un mensaje al usuario indicando cuál es el error y que debe corregir. El botón “Limpiar” borra los campos dirección MAC y Nombre para que el usuario pueda volver a introducir nuevamente la información.

Para finalizar, el botón “Aceptar” abre la ventana de visualización de datos (figura 25) siempre y cuando haya al menos un dispositivo asociado. En caso contrario se muestra un mensaje informativo indicando al usuario que debe existir al menos un dispositivo asociado.

#### 4.3.3.3. Pantalla de visualización de datos

La pantalla de visualización de datos (figura 25) el usuario podrá seleccionar, de una lista seleccionable, el dispositivo sobre el que podrá visualizar los datos del sensor. Además, en esta pantalla el usuario podrá volver a la configuración de dispositivos utilizando el botón “Configurar” y salir de la aplicación pulsando el botón “Salir”. Además, en esta pantalla el usuario puede visualizar el estado de la calidad del aire en modo texto. En ella se indica que la calidad del aire puede ser “Normal” o “Deficiente” si supera el umbral definido, y también se representa en modo gráfico que cambia de color si supera dicho umbral. Por último, el usuario puede cambiar este umbral de modo que la aplicación lo utilizará para cambiar el comportamiento y mostrar alertas, textos y colores si alguno de los sensores supera dicho umbral.

***Figura 25.*** *Prototipo de la pantalla de visualización de datos.*


Fuente: Elaboración propia, 2021.

## 4.4. Desarrollo de la solución

En este apartado de la memoria se detallarán las funciones principales del código fuente de los distintos componentes que forman parte de la solución, la APP, los microcontroladores ESP32 y ESP8266 y el sensor de gas MQ-135. Para más detalle se puede consultar el repositorio completo en GitHub[4] a través de la URL: https://github.com/absoca/control_CO_APP

El repositorio está constituido por tres directorios, el directorio "App Inventor" que incluirá el fichero .aia dónde se podrá revisar el código fuente de la aplicación móvil, el directorio "ESP32" que incluirá los ficheros fuentes cargados en el microcontrolador y el directorio "ESP8266" que incluirá los ficheros fuentes correspondientes a este dispositivo.

[4] Plataforma de desarrollo colaborativo software que permite alojar los códigos fuente de aplicaciones utilizando control de versiones.

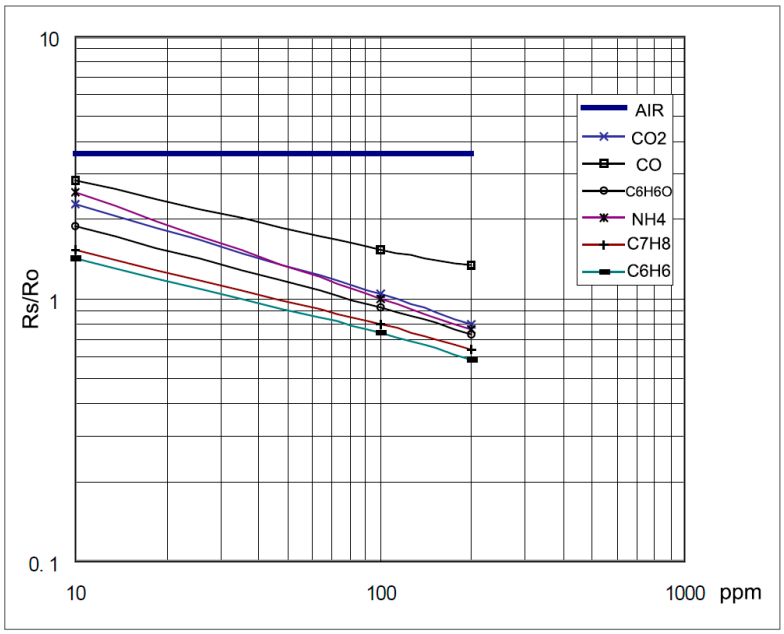
### 4.4.1. Funciones principales del ESP8266

#### 4.4.1.1. Interpretación de datos del sensor MQ-135

El dispositivo MQ-135 dispone de dos salidas para poder interpretar los resultados, una salida digital y otra analógica. La respuesta de la salida digital puede ser 0 o 1 en función de la sensibilidad establecida a través del potenciómetro. La respuesta analógica puede tomar valores de 0 a 1023 según la concentración del gas que se quiera medir. En nuestro caso interesa obtener el valor analógico para representarlo gráficamente y compararlo con valor que es el umbral.

Para la obtención de datos de manera analógica no basta con simplemente leer el valor proporcionado por el sensor, si no que debemos transformarlo según el gas que se esté midiendo. Para ello el fabricante proporciona una gráfica con las distintas respuestas del sensor según el gas (figura 26). Estas medidas han sido realizadas bajo unas ciertas condiciones de humedad (65%), de temperatura (20º C) y concentración de oxígeno (21%).

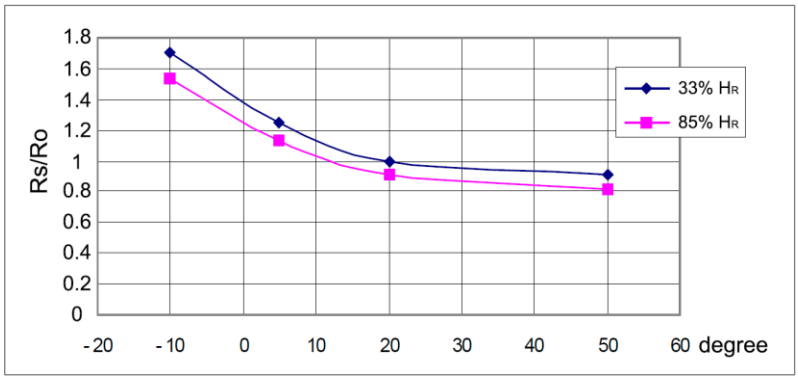
***Figura 26.*** *Sensibilidad del sensor MQ-135 según el tipo de gas.*




Fuente: AZ-Delivery, 2021.

El fabricante hace especial hincapié sobre la dependencia de los resultados del sensor causados por la humedad y la temperatura (figura 27). En nuestro caso el sensor estará situado dentro de una casa donde la temperatura debería oscilar a unos 20-25º C durante el día y los 15-17º C por la noche y la humedad relativa debería oscilar entre el 35-65%. Teniendo en cuenta las curvas azul y magenta que representan la humedad relativa ($H_R$) entre el 33 y el 85% y el rango de temperaturas que estará el sensor (15-25º C) el factor de corrección es prácticamente 1. En este sentido no se abordará en este proyecto el autoajuste del sensor en función de la humedad y la temperatura.

***Figura 27.*** *Ajuste del sensor MQ-135 según la temperatura y humedad.*




Fuente: AZ-Delivery, 2021.

Para obtener la ecuación que nos permita obtener la concentración, dado que las escalas de ambos ejes de la gráfica mostrada en la figura 26 son logarítmicas, consideramos que en general que el comportamiento de la concentración es una recta bajo estos ejes. De este modo podemos simplificar la fórmula a:

$$Concentración = 10^{(A*\log\left(\frac{Rs}{R}\right)+B)}$$

Para determinar la concentración será necesario obtener la recta que la aproxima, de modo que elegiremos dos puntos cuales quiera de la gráfica representada en la figura 26 P0=(X0, Y0) y P1=(X1, Y1), considerando la ecuación de la recta:

$$Y = A * x + B$$

Dónde:

$$A = \frac{Y_1 - Y_0}{X_1 - X_0}$$

$$B = Y_0 - A * X_0$$

De modo que, teniendo en cuenta las condiciones iniciales de temperatura y la aproximación a una recta en nuestro código podemos establecer las siguientes variables (figura 28):

***Figura 28.*** *Variables de entorno para calcular la concentración.*

```
// Datos para la media en la lectura de múltiples valores
const int TIEMPO_MUESTRAS = 250;
const int NUMERO_MUESTRAS = 10;

//Valores de las resistencias para el gas CO
const int RL = 20;      //Resistencia del modulo en Kilo ohms
const int R0 = 3.7;          // Resistencia del sensor en Kilo ohms

// Valores de la recta según el Datasheet y el gas CO
const float X0 = 10;
const float Y0 = 1.9;
const float X1 = 200;
const float Y1 = 1.4;

// Puntos de la curva de concentración {X, Y}
const float punto0[] = { log10(X0), log10(Y0) };
const float punto1[] = { log10(X1), log10(Y1) };

// Calcular pendiente y coordenada abscisas
const float scope = (punto1[1] - punto0[1]) / (punto1[0] - punto0[0]);
const float coord = punto0[1] - punto0[0] * scope;
```

Fuente: Elaboración propia, 2021.

El siguiente paso es implementar las funciones que realizarán la media de las lecturas y obtención de la resistencia del sensor ($R_S$) para posteriormente obtener la concentración. La lógica de la primera función es sencilla, un pequeño bucle iterativo "*for*" con un número configurable de iteraciones donde acumula la lectura del sensor y realiza la media.

***Figura 29.*** *Cálculo de promedio de muestras y obtención de la resistencia del sensor.*

```
// Obtener la resistencia promedio en N muestras
float leerSensor(int pin_sensor)
{
   float rs = 0;
   for (int i = 0;i<NUMERO_MUESTRAS;i++) {
      rs += obtenerResistencia(analogRead(pin_sensor));
      delay(TIEMPO_MUESTRAS);
   }
   return rs / NUMERO_MUESTRAS;
}

// Obtener resistencia a partir de la lectura analogica
float obtenerResistencia(int lectura_sensor)
{
   return ((float)RL / 100.0*(1023 - lectura_sensor) / lectura_sensor);
}
```

Fuente: Elaboración propia, 2021.

Para finalizar, la función final de obtención de la concentración tiene como entada el valor obtenido con la función *leerSensor()* que ha sido el promedio de N muestras. En esta función se ha implementado la fórmula del cálculo de la concentración.

***Figura 30.*** *Cálculo de la concentración.*

```
// Obtener concentracion 10^(A * log (rs/r0) + B)
float obtenerConcentracion(float rs_media)
{
   float ratio = rs_media/R0;
   return pow(10, scope * log(ratio) + coord);
}
```

Fuente: Elaboración propia, 2021.

#### 4.4.1.2. Configuración del protocolo ESP-NOW

Otro punto importante a destacar dentro del código fuente del ESP8266 es la conectividad mediante el protocolo ESP-NOW. Para empezar a utilizar las funciones básicas de este protocolo es necesario añadir las librerías *ESP8266WiFi.h* y la *espnow.h*. La primera librería únicamente se necesita para obtener la dirección MAC del dispositivo. Respecto la segunda librería, es necesaria incluirla para poder establecer el rol del dispositivo dentro de la red (maestro, esclavo o combinación de ambos), la función de callback para el envío y para la

respuesta y finalmente realizar el emparejado. Esta inicialización deberá realizarse en la función *setup()* del microcontrolador (figura 31).

***Figura 31.*** *Inicialización de variables para ESP-NOW.*

```
void setup() {
  Serial.begin(9600);
  Serial1.begin(9600);

  Serial.print("ESP8266 Board MAC Address:  ");
  Serial.println(WiFi.macAddress());

  // Set device as a Wi-Fi Station
  WiFi.mode(WIFI_STA);

  // Iniciamos ESP-NOW
  if (esp_now_init() != 0) {
    Serial.println("Error initializing ESP-NOW");
    return;
  }

  // Despues de la inicialización de ESPNow establecemos el rol
  esp_now_set_self_role(ESP_NOW_ROLE_COMBO);
  esp_now_register_send_cb(OnDataSent);
  esp_now_register_recv_cb(OnDataRecv);

  // Registramos el emparejado
  esp_now_add_peer(broadcastAddress, ESP_NOW_ROLE_SLAVE, 1, NULL, 0);
}
```

Fuente: Elaboración propia, 2021.

Las funciones de callback *OnDataSent* y *OnDataRecv* serán las encargadas de ejecutarse cuando se envíe y reciba un evento respectivamente. La función de envío solamente indica por puerto serie que el paquete de datos ha sido enviado correctamente o ha habido un fallo en la entrega. Respecto la segunda función es la encargada de recibir el paquete de datos e invocar a la función de *enviarDatosSerializados()*. Esta última función es la encargada de enviar por el puerto serie 1 del ESP8266 el string de datos en formato JSON, por ejemplo, *{"MAC":"11:22:33:44:55:66","data":523}*.

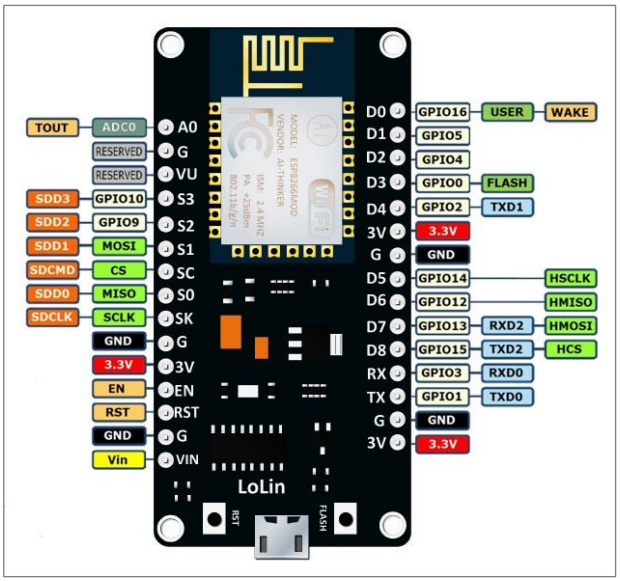
Para realizar el envío de datos al resto de dispositivos se añade en la función principal del programa la llamada al método *esp_now_send()* pasándole como parámetros la dirección MAC, los datos y el tamaño en bytes de los datos enviados. Como la idea de este proyecto es que asociar y desasociar dispositivos sea transparente para el usuario final, ESP-NOW dispone

de la funcionalidad de broadcast. Esta funcionalidad solamente necesita pasarle al método de enviar la dirección MAC como FF:FF:FF:FF:FF:FF, de este modo puede enviar y recibir de cualquier otro dispositivo.

#### 4.4.1.3. Envío de datos por el puerto serie

El microcontrolador ESP8266 dispone de tres puertos series, el 0, el 1 y el 2. Aunque se pueden utilizar todos los puertos serie, en este proyecto se ha decidido dejar el puerto serie 0 como el puerto para mostrar mensajes de debug y utilizar el puerto serie 1 como envío de datos ya que además se corresponde con el GPIO2 del ESP8266 que está mapeado internamente con el led azul que hay en la placa. De este modo cada vez que se envíen datos por el puerto se encenderá el led. Cuando hagamos la conexión física deberemos tener en cuenta, siguiendo el diagrama de pinout[5] de la placa, que se corresponde con el pin señalado como D4 (figura 32).

***Figura 32.*** *Pinout del módulo NodeMCU v3 con el ESP8266 integrado.*




Fuente: Az-Delivery, 2021.

[5] Es un término anglosajón utilizado comúnmente en electrónica para indicar la correspondencia entre los pines físicos de la placa de desarrollo y los pines de entrada y salida del dispositivo SoC (en este caso el ESP8266).

La función de envío de datos por el puerto serie es muy sencilla, solamente es necesario inicializar el puerto y la velocidad de transmisión en la función *setup()* del dispositivo y después enviar datos usando el método *write* (envío byte a byte) o *print* (envío de cadenas de caracteres). Como protocolo de comunicación aprovecharemos la nomenclatura JSON para indicar que la transmisión del string empieza por el carácter { y termina con el carácter } (figura 33). Después, en el apartado de procesamiento de los datos recibidos en el ESP32, se detallará la lógica de control que debe tener el dispositivo para aceptar o rechazar la recepción de los datos.

***Figura 33.*** *Función para enviar datos por el puerto serie 1.*

```
void enviarDatosSerializados(){
  Serial.println("Envío trama por serial 1");
  //Enviamos los datos del sensor remoto recibido
  Serial1.print("{\"MAC\":\"");
  Serial1.print(dataRecv.mac);
  Serial1.print("\",\"data\":");
  Serial1.print(dataRecv.valor);
  Serial1.print("}");
}
```

Fuente: Elaboración propia, 2021.

### 4.4.2. Funciones principales del ESP32

#### 4.4.2.1. Creación del punto de acceso e inicialización del servidor

Para dotar al ESP32 de la funcionalidad de punto de acceso y servidor web es necesario incluir dos librerías, la librería *WiFi.h* y la librería *WebServer.h*. La primera de ellas te permite levantar un punto de acceso con un SSID y contraseña con el método *softAP()*. Para la inicialización del server debe crearse un objeto *WebServer* que escucha por el puerto HTTP 80 para posteriormente, en la función *setup()* del programa principal realizar la invocación al método *begin()* del servidor para arrancarlo. En el siguiente apartado se detallará la exposición de los métodos HTTP que es necesario inicializarlos antes de arrancar el servidor.

#### 4.4.2.2. Exposición de métodos HTTP

Como se comentaba anteriormente, antes de arrancar el servidor web es necesario exponer los métodos sobre los que el propio servidor responderá cuando reciba peticiones de clientes conectados al punto de acceso. Estas declaraciones deben realizarse en el *setup()* del microcontrolador. En el proyecto del ESP32 se creó una librería *Server.hpp* que contiene todas las funciones relativas a la comunicación del propio servidor.

Como primera función de esta librería podemos destacar la *initServer()* dónde es la responsable de exponer todas las rutas, los métodos sobre los que deben responder y la función que se ejecutará cuando reciban la petición. Una vez establecidos los métodos, se arranca el servidor con el método *begin()* (figura 34).

***Figura 34.*** *Función initServer() que establece las rutas HTTP del servidor.*

```
////////////////INICIALIZACIÓN//////////////////////////
void initServer()
{
  dispEncontrados=0;
////////////////////RUTEO GENÉRICO//////////////////////////
  // Ruteo para '/'
  server.on("/", handleRoot);

  // Ruteo para URI desconocida
  server.onNotFound(handleNotFound);

////////////////////RUTEO GET//////////////////////////
  server.on("/values", HTTP_GET, handleGetValues);

////////////////////RUTEO POST//////////////////////////
  // Ruteo para recibir las MAC por POST
  server.on("/mac", HTTP_POST, handleSetMAC);

////////////////////INICIO SERVER//////////////////////////
 server.begin();
 Serial.println("HTTP server started");
}
```

Fuente: Elaboración propia, 2021.

Como se puede observar en la figura 34 el servidor responderá bajo 3 rutas, la ruta */* que mostrará un mensaje de bienvenida, la ruta */values* que devolverá los valores de los

sensores y la ruta */mac* que inicializará los dispositivos que se ha pasado por parámetro. Esta última llamada, al tratarse de envío de datos, se realiza por POST. Si se intenta acceder a otra URL que no se corresponda con alguna de las 3 anteriormente citadas, se ejecutará el ruteo para URL no encontrada devolviendo un código de retorno HTTP 404 (página no encontrada).

Cuando se expone la ruta y el método también debe especificarse la función de callback que será llamada una vez reciba el servidor la petición a la URL específica. En este caso se han definido una para cada URL expuesta, la función *handleRoot* que responderá bajo la ruta por defecto o */*, la función *handleGetValues* (figura 35) que responderá bajo el path */values* y la función *handleSetMAC* que responderá bajo la petición */mac*.

***Figura 35.*** *Función handleGetValues que devuelve los valores de los sensores.*

```
// Funcion que se ejecutara en URI /values
  void handleGetValues()
  {

    //Creamos el documento JSON
    DynamicJsonDocument doc(capacidadJSON);

    DeserializationError error = deserializeJson(doc, dataSensores);
    //Comprobamos si no tenemos ningún error con el formato del JSON
    if (error) {
      server.send (500, "text/json", "[{\"type_error\": 1,\"cod_error\":"
      + (String)error.code()
      + ",\"desc_error\":\""
      + (String)error.c_str()
      +"\"}]");
    }
    else
      server.send (200, "text/json", dataSensores);
  }
```

Fuente: Elaboración propia, 2021.

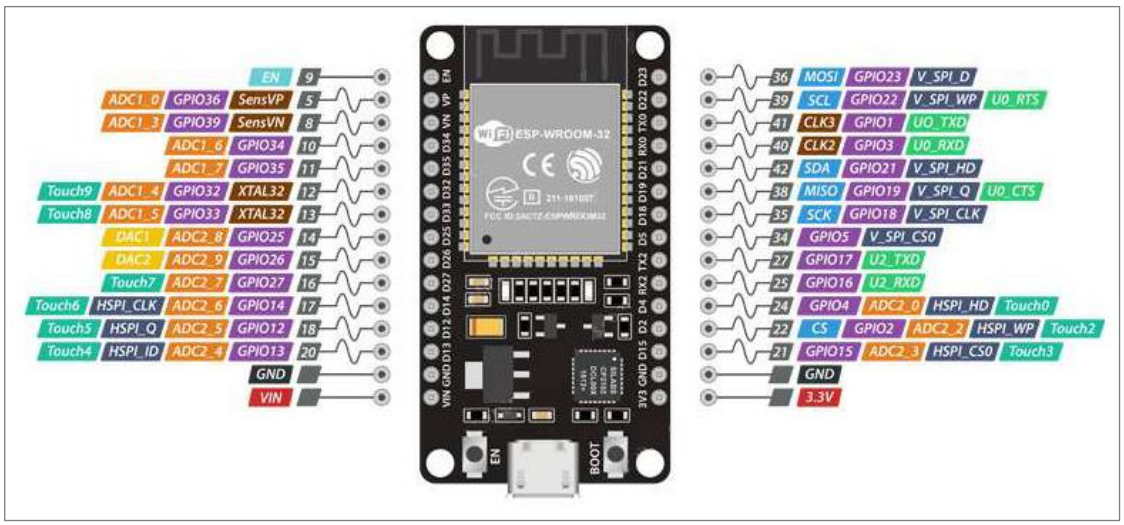
En estas funciones, después de realizar las operaciones necesarias, deberá indicarse el tipo de respuesta que se desea devolver. Como se puede ver en la figura 35 podemos establecer respuestas distintas en función de los resultados que podamos obtener durante la ejecución del código, en este caso un error HTTP 500 o un código HTTP 200 si ha ido todo bien. Además de especificar el código HTTP es necesario indicar el MIME type, en este caso concreto como el formato de las respuestas son JSON, el MIME type debe ser "*text/json*". Después del

tipo de datos que devolverá la petición HTTP se pasará la información a devolver, o bien un texto estático como un string o bien el valor de una variable.

#### 4.4.2.3. Recuperación de datos por el puerto serie

Como se ha comentado anteriormente, la comunicación entre el ESP32 y el ESP8266 se realizará a través de los UART disponibles en los microcontroladores. Al igual que el ESP2866, el ESP32 dispone de tres puertos series aunque depende de la placa de desarrollo que se utilice el puerto serie 1 puede no estar accesible. En nuestro caso al utilizar la placa de desarrollo de 30 pines solamente están los puertos series 0 y 2 (figura 36). En este proyecto solamente se utilizará el pin RX2 del puerto serie 2 para recibir la información del ESP8266.

***Figura 36.*** *Pinout del ESP32 WROOM-32 de Espressif.*




Fuente: Circuits4you, 2021.

Del mismo modo que todas las funciones relativas al servidor se independizaron en una librería, todas las funciones relativas a la comunicación por el puerto serie se han agrupado en una librería llamada *Comunicacion.hpp*. También está definida la función *initComunicacion()* correspondiente a la inicialización de variables relativas a la comunicación como son los flags de control *tramaCompleta*, *inicioTrama*, *finTrama* y *cadena*.

La función principal de esta librería es *leerSerial()* encargada, como su propio nombre indica, de leer los datos que recibe por el puerto serie. Esta función se llamará desde la función *loop()* con un retardo para solicitar datos cada cierto tiempo. Esta función serial, antes de empezar a leer los bytes comprueba si el puerto está disponible con la función *available()*, en caso que no esté disponible, no ejecuta nada y devuelve vacío.

El siguiente paso, tras comprobar que el puerto serie 2 está disponible, es comprobar si la trama recibida está completa, en caso afirmativo comprueba si los flags de control de inicio y fin de trama están a *true*, si ambos están a *true* significa que la trama recibida está correcta y se devuelve la cadena para que se procese en formato JSON posteriormente en la memoria del ESP32.

Aprovechando el formato que nos brinda JSON, se ha considerado que el inicio de trama lo marcará la llave de apertura *{* y el fin de trama lo marcará la llave de cerrado *}*. Cuando se detecta la llave de apertura, se cambia el flag de inicio de trama a *true* y del mismo modo, cuando se recibe la llave de cerrado se establece el flag de fin de trama a *true* (figura 37).

***Figura 37.*** *Comprobación del inicio y final de trama.*

```
inByte = Serial2.read();
if (inByte =='{'){
  cadena=""; //Recibimos el caracter de inicio de trama y borramos el string
  tramaCompleta=false;
  inicioTrama=true;
  finTrama=false;
} else if (inByte =='}'){
  finTrama=true;
  if (inicioTrama){
    tramaCompleta=true;
    }
  else
    finTrama=false;
}
```

Fuente: Elaboración propia, 2021.

Para la lectura por el puerto serie se ha utilizado el método *read()*. Con este método se obtiene byte a byte para su posterior análisis de cada carácter y concatenarlo al string final. En este punto, antes de seguir con la concatenación, como la comunicación puede sufrir ruidos o variaciones que puedan romper la cadena, se comprueba que la longitud de la cadena tiene un tamaño máximo definido, en caso de rebasar este umbral, se resetean todos los flags y se borra la cadena esperando nuevamente una comunicación válida (figura 38) y desestimando los datos recibidos.

***Figura 38.*** *Comprobación de tamaño de trama para protección contra errores.*

```
if(inicioTrama){
  //Comprobamos si la cadena obtenida tiene mayor longitud de la esperada
  if (cadena.length() > LON_CADENA){
    inicioTrama=false;
    finTrama=false;
    tramaCompleta=false;
    break;
  } else
    cadena+=String(inByte); //Construyo la cadena caracter a caracter concatenándolos
}
```

Fuente: Elaboración propia, 2021.

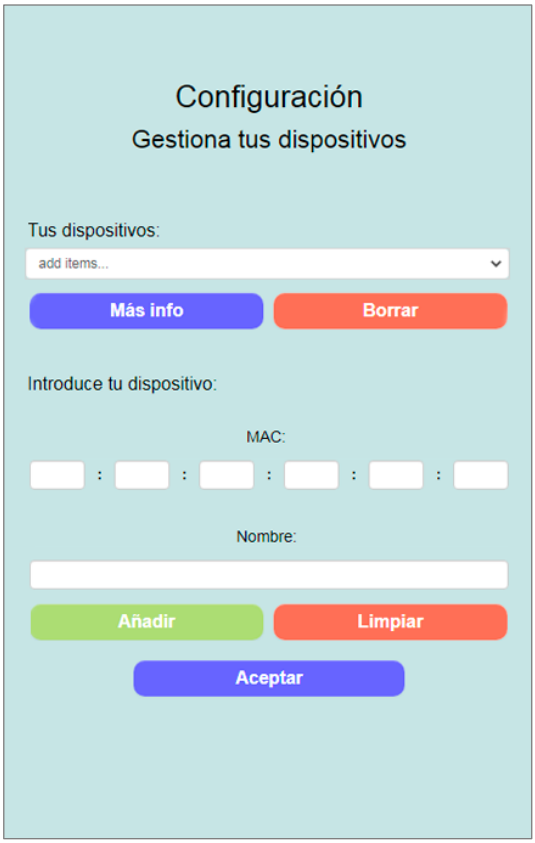
### 4.4.3. Funciones principales de la APP

Otro punto de implementación de código en este proyecto ha sido la creación de la aplicación móvil para dispositivos Android. Para ello se ha utilizado el IDE de desarrollo App Inventor que ofrece un interfaz sencillo, basado en bloques, para desarrollar pequeñas APPs.

#### 4.4.3.1. Administración de dispositivos

La pantalla de administración de dispositivos permite al usuario añadir, modificar o borrar los dispositivos sobre los cuales necesita recibir la información. Estos dispositivos están identificados por su dirección MAC como clave única y por un nombre para su fácil identificación (asignando nombres como cocina, lavandería, baño, etc.).

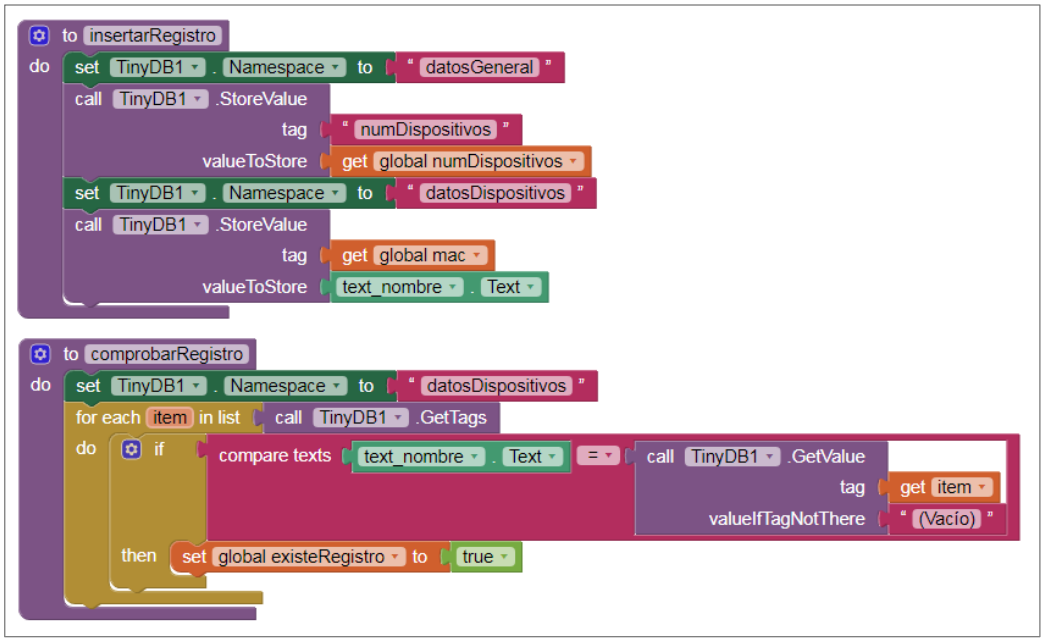
***Figura 39.*** *Pantalla de configuración de dispositivos.*




Fuente: Elaboración propia, 2021.

En esta pantalla (figura 39) se ha insertado el componente de almacenamiento de datos persistentes TinyDB que dispone App Inventor para almacenar los distintos dispositivos y disponer de dicha información en otras pantallas como la de visualización de datos. En la figura 40 se puede comprobar los procedimientos *comprobarRegistro* e *insertarRegistro*. El primero se encarga de comprobar si el dispositivo que se intenta guardar ya existe, en caso afirmativo establece la variable de control *existeRegistro* a *true*. Si en el flujo de guardado de información el programa detecta que esa variable tiene el valor a *false*, llama al procedimiento *insertarRegistro* para realizar la inserción, en caso contrario muestra al usuario una notificación indicando que el dispositivo ya existe. En ambos métodos, como se puede apreciar, antes de realizar la consulta hacia la tabla se debe establecer el namespace concreto, en este caso se corresponde con el *datosDispositivos*.

***Figura 40.*** *Procedimientos de inserción y chequeo de registros en base de datos.*




Fuente: App Inventor, 2021.

Además de la comprobación en base de datos si existe el dispositivo, antes de llamar a estos procedimientos, también existe un chequeo previo de los campos de entrada. Los procedimientos creados han sido *comprobarCamposVacios*, *comprobarCamposVaciosMAC, comprobarCamposMayoresNombre*, *comprobarCamposMayoresMAC* y *validarMAC*. Los procedimientos de comprobación de campos vacíos verifican que el campo introducido no está vacío (el nombre del dispositivo o la dirección MAC), en caso que alguno esté vacío, o ambos, se colorean los campos de texto en color amarillo y además se muestra un mensaje informativo indicando que debe revisar los campos vacíos.

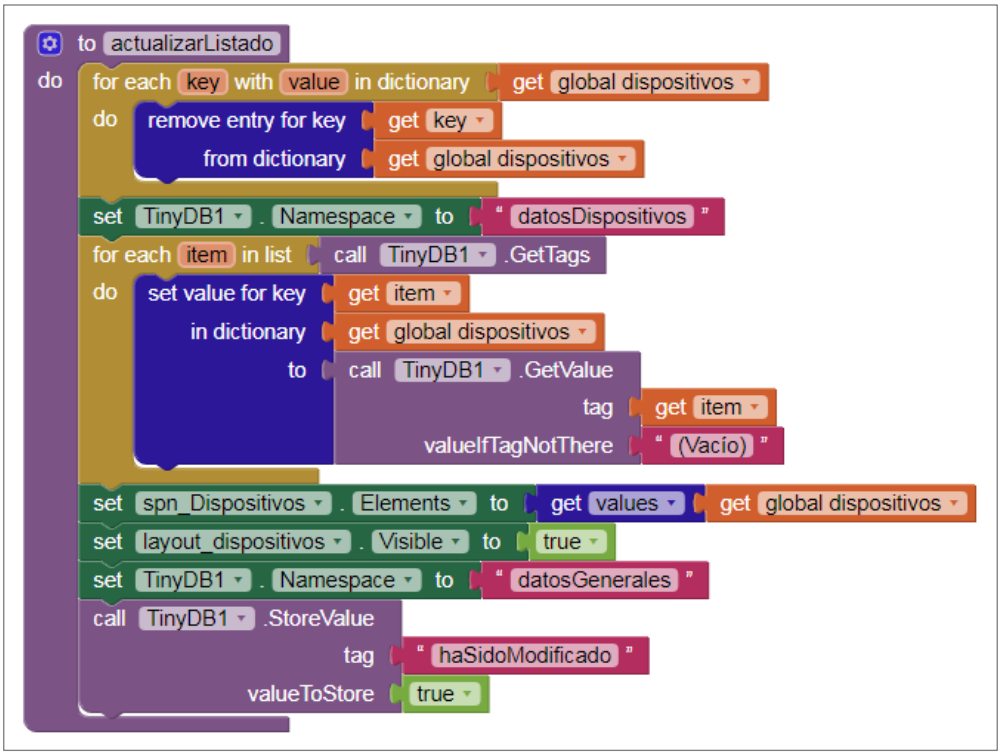
En referencia a los otros dos procedimientos, su tarea es comprobar el número de caracteres de cada campo. Para el caso del nombre se ha establecido un número máximo de caracteres que ha sido 15, en el caso de la dirección MAC, la longitud de cada campo debe ser 2.

Por último, el procedimiento *validarMAC* se encarga de comprobar que los caracteres introducidos están dentro del rango aceptado de valores hexadecimales, es decir, de 0 a 9 y de A a F. En caso de no cumplir esta validación se muestra el mensaje informativo.

Por otra parte, cabe destacar el procedimiento encargado de mantener actualizado el desplegable dónde aparecen los dispositivos guardados. El procedimiento *actualizarListado* (figura 41) primero reinicia la variable de tipo diccionario que almacena la dupla dirección MAC y nombre. Una vez reiniciada esta variable se establece el namespace correcto y se obtienen todos los registros y por cada registro obtenido, se guarda nuevamente en el diccionario. El motivo por el cual se ha decidido utilizar este tipo de variable ha sido por simplicidad con el manejo de la información y simplicidad para almacenar los datos en la lista desplegable.

Tras el almacenamiento correcto se actualizan los campos de la lista desplegable y se guarda en una variable global el flag de control *haSidoModificado* con valor *true* que indica que el usuario ha realizado una modificación sobre los dispositivos y en la siguiente pantalla, visualización de datos, deberá tenerse en cuenta para indicar al servidor que el listado ha sufrido un cambio y actualizarse en consecuencia.

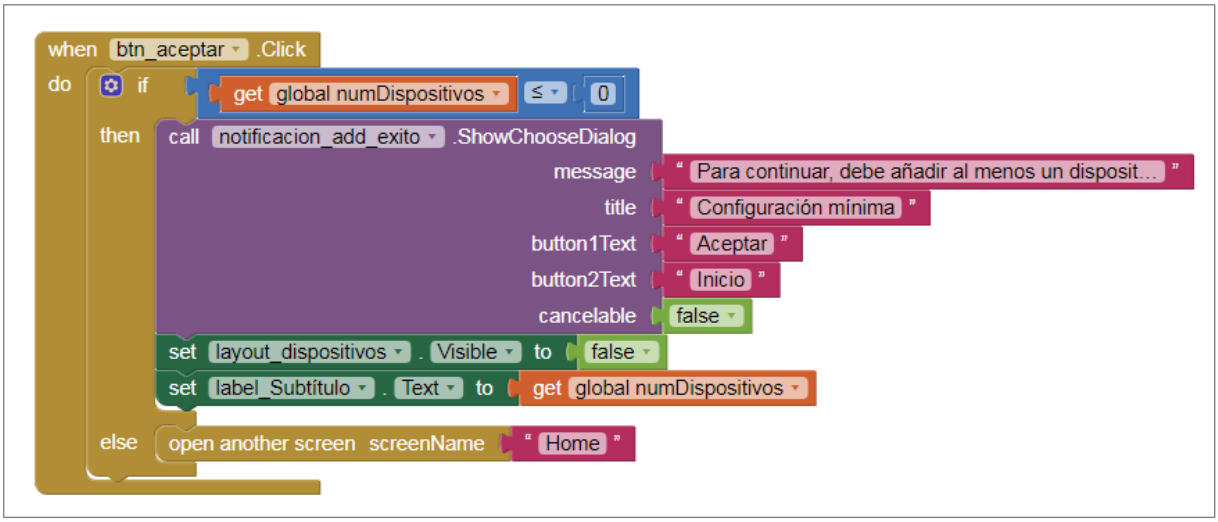
***Figura 41.*** *Procedimiento encargado de actualizar el desplegable de dispositivos.*




Fuente: App Inventor, 2021.

Para finalizar, el usuario debe pulsar el botón “Aceptar” que le permite salir de la página de configuración. En este punto se controla si el número de dispositivos es menor o igual a 0, en caso afirmativo se muestra un mensaje informativo (figura 42) indicando que no es posible acceder a la pantalla de visualización de datos y ofreciéndole la posibilidad de volver a la página anterior para insertar un dispositivo o volver a la página de inicio. En el caso que el número de dispositivos sea mayor a 0, utilizando el método de abrir ventana nueva de App Inventor, se abre la ventana de visualización de datos.

***Figura 42.*** *Manejo del evento click del botón aceptar.*




Fuente: App Inventor, 2021.

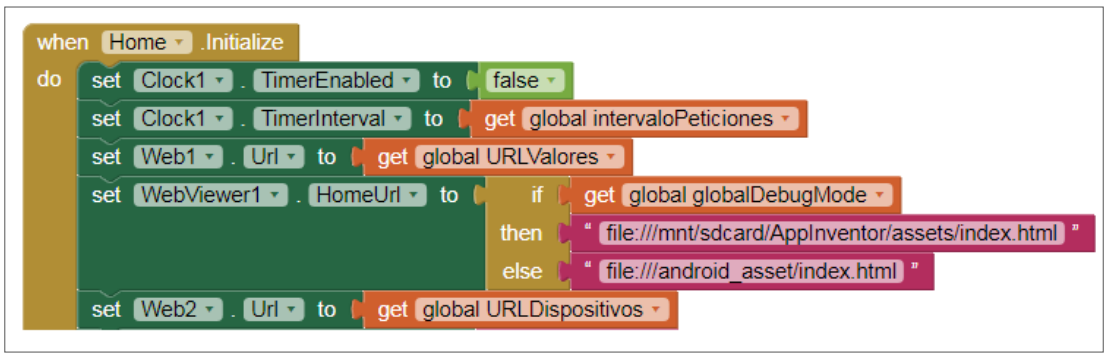
### 4.4.3.2. Recuperación y visualización de datos

La pantalla de visualización de datos es la encargada de mostrar la información de manera gráfica y textual de los valores recuperados de los distintos sensores asociados a los dispositivos. Esta pantalla tiene cuatro módulos principales que son el componente webview, el componente web, el componente clock y el componente de notificaciones push.

El componente webview se ha utilizado para mejorar la funcionalidad de representación de datos de App Inventor. El IDE de desarrollo dispone de componentes visuales limitados para la representación de datos como pueden ser gráficas de barras, gráficas circulares, etc. Para ello, dentro del apartado *Media* de App Inventor se han insertado

previamente una página web HTML con una referencia a la libería javascript Chart.js que, tras recibir por parámetros del webview un JSON, representa una gráfica de tipo donut o circular. Para hacer uso de este recurso web se ha utilizado el componente webview en el que se ha establecido esta página HTML como página de inicio (figura 43).

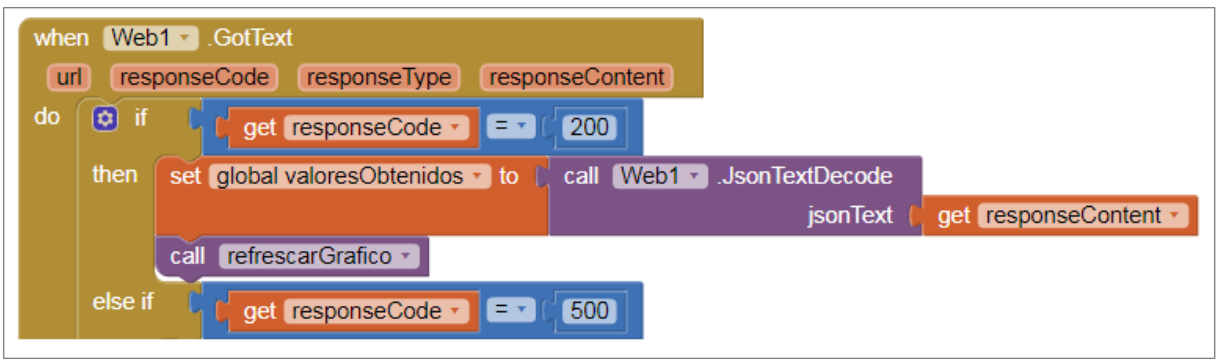
***Figura 43.*** *Uso del componente webview en App Inventor.*




Fuente: App Inventor, 2021.

Como se puede apreciar en la figura 43, antes de llamar al componente webview, se invoca al componente web con una URL. Esta URL es la que se expuso en el servidor web para obtener los valores de los dispositivos, el path */values*. El componente web dispone de un manejador de eventos cuando recibe una respuesta tras la invocación a la URL solicitada. Este evento expone cuatro variables que te permite controlar la URL solicitada, el código HTTP de respuesta, el MIME type y el contenido de la respuesta HTTP (figura 44).

***Figura 44.*** *Sección del manejo de la respuesta HTTP del componente web.*



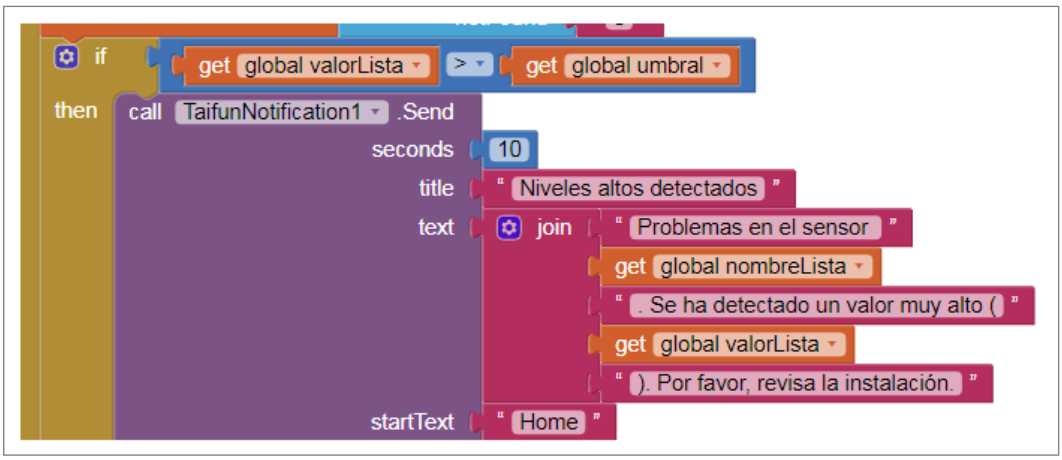

Fuente: App Inventor, 2021.

Si el código de respuesta HTTP indica que es un 200, respuesta correcta, se procede a la invocación del procedimiento *refrescarGrafico* encargado de procesar la respuesta. Esta respuesta previamente es almacenada en una variable global que, gracias al componente de interpretación de objetos JSON de App Inventor, se puede recuperar fácilmente.

Este procedimiento es el encargado de interpretar los datos recibidos por la petición */values* que por una parte controla si debe mostrar o no la notificación push, por otra parte debe actualizar el texto informativo de la calidad del aire y finalmente componer el JSON que posteriormente se enviará al webview.

Gracias al uso de extensiones de App Inventor que permite ampliar la funcionalidad de este IDE de desarrollo, se ha utilizado la extensión *TaifunNotification* implementada por Taifun (2021) que facilita el uso de las notificaciones push. En este sentido solamente fue necesario añadir el componente a la aplicación y posteriormente invocarlo con los parámetros adecuados como son el número de segundos que debe permanecer la notificación, el título, el texto a mostrar y el texto que se le pasará a la ventana inicial de la APP (figura 45). Este texto inicial te permite establecer un control en la aplicación sobre qué ventana abrir ya que por defecto abre la pantalla inicial.

***Figura 45.*** *Uso del componente de notificación push.*

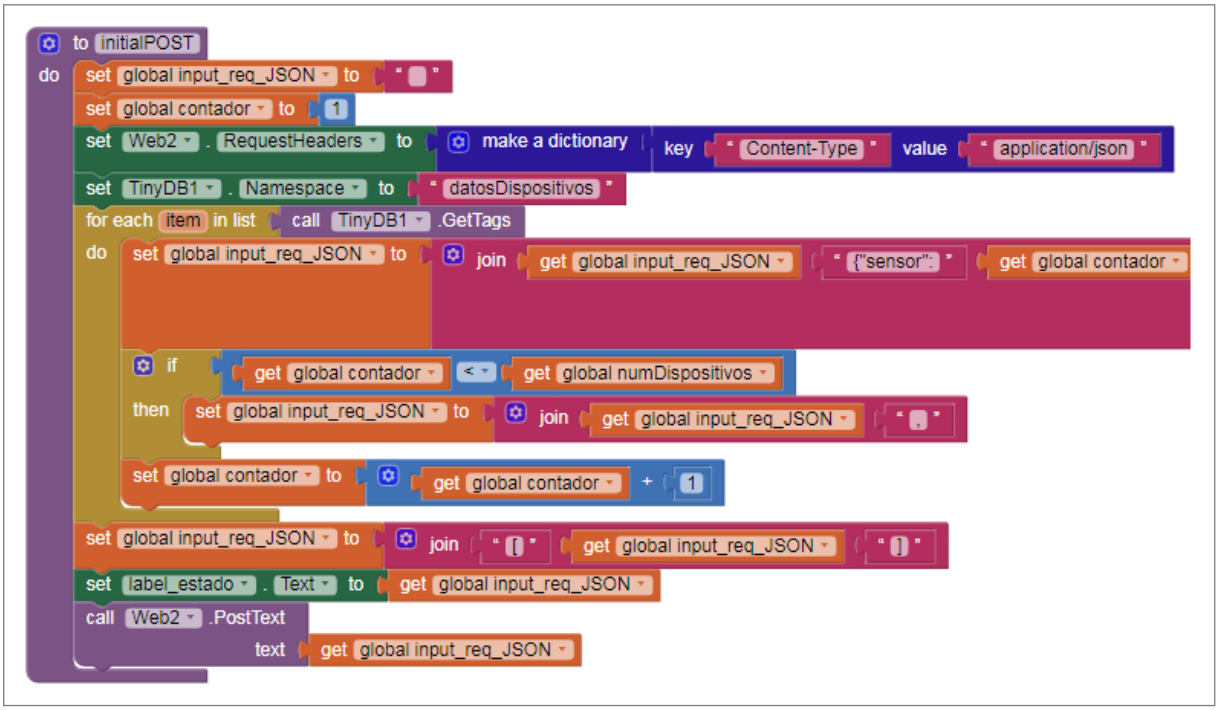



Fuente: App Inventor, 2021.

Como hemos enumerado anteriormente, dentro de los componentes destacables de esta pantalla, se encuentran los componentes web que permiten realizar peticiones HTTP hacia un recurso web. App Inventor permite realizar los principales métodos HTTP, GET, POST, PUT y DELETE. En esta ocasión solamente se ha necesitado utilizar el método GET para obtener los valores de los sensores y el método POST para establecer los dispositivos en el servidor web encargado de recuperar la información de los sensores.

Esta llamada POST se ejecuta bajo las siguiente condiciones, cuando la llamada de obtención de valores devuelve un error controlado indicando que no existe la estructura de dispositivos en el servidor web o cuando el usuario modifica el listado de dispositivos desde la pantalla de configuración. Esta llamada POST está en capsulada dentro de un procedimiento específico llamado *initialPOST (figura 46)*.

***Figura 46.*** *Sección del procedimiento para ejecutar una petición HTTP mediante POST.*



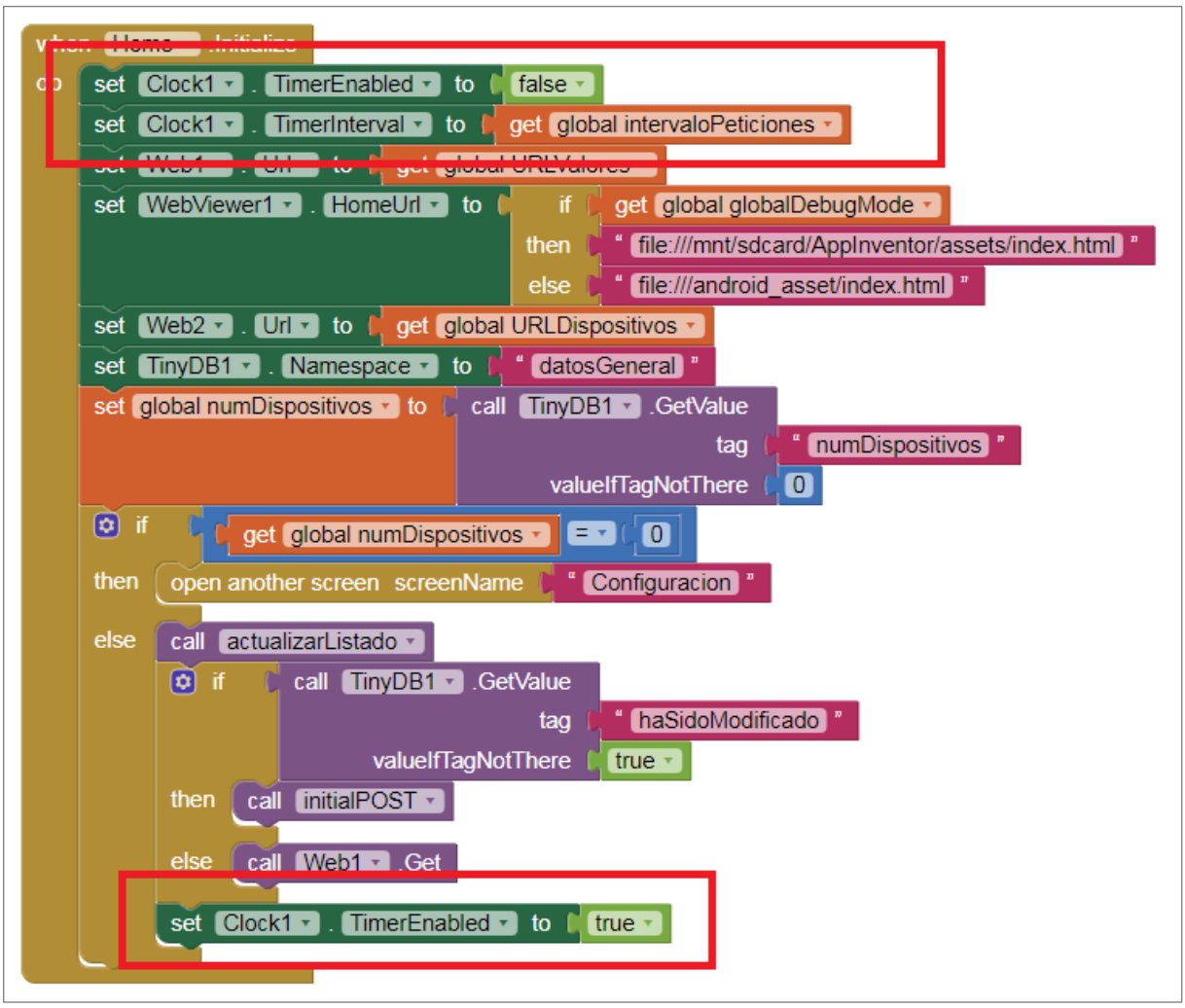

Fuente: App Inventor, 2021.

Dicho procedimiento primeramente se encarga primeramente de recuperar de la base de datos interna de la aplicación los dispositivos. Este listado de dispositivos se almacena en

una variable global *input_req_JSON* que contiene un string con un JSON. Posteriormente, tras formar el JSON se produce la llamada al componente web pasándole como parámetro la variable que contiene el JSON con los nombres y direcciones MAC de los dispositivos insertados por el usuario.

Para finalizar, el último componente destacable de esta pantalla es el clock. Este módulo permite disponer de un temporizador que puede lanzar eventos timer cada cierto tiempo. En este proyecto los eventos timer se han utilizado para realizar polling de modo que cada cierto tiempo la aplicación pueda ejecutar peticiones HTTP de manera automática y refrescar los valores de los sensores en la aplicación.

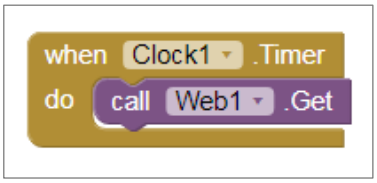
***Figura 47.*** *Inicialización del componente clock.*




Fuente: App Inventor, 2021.

Como se puede observar en la figura 47, cada vez que la pantalla de visualización de datos se inicializa, se bloquea el componente clock, se establece el intervalo que provocará los eventos del timer y tras comprobar que una primera llamada a la obtención de valores es correcta, se habilita nuevamente el timer. Después debe capturarse el evento timer e indicar qué serie de operaciones debe ejecutar, en este caso realizar una llamada HTTP por método GET (figura 48).

***Figura 48.*** *Manejador de eventos timer del componente clock.*




Fuente: App Inventor, 2021.

4.4.3.3. Selección y conexión del punto de acceso

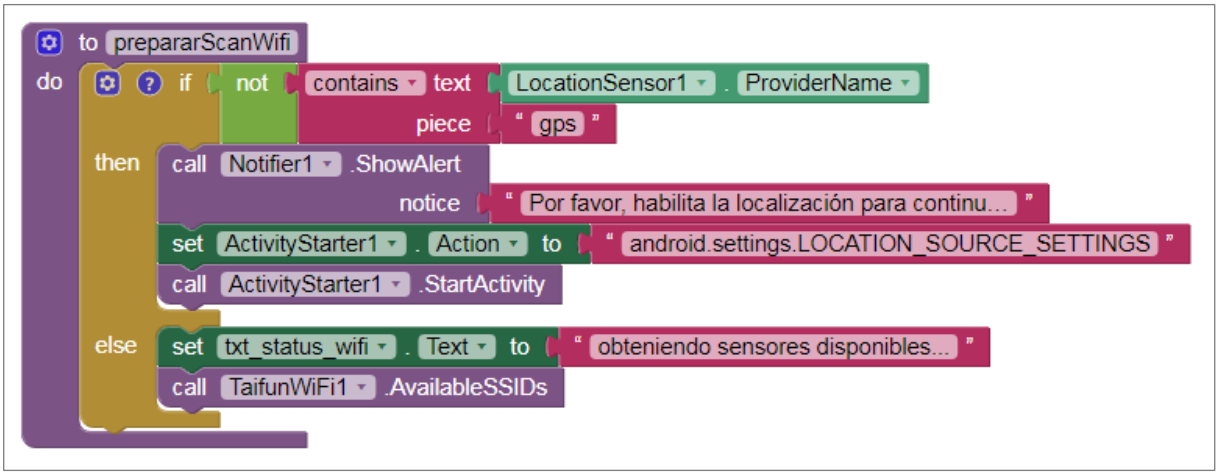
El último punto remarcable del desarrollo con App Inventor es la conexión con el punto de acceso creado con el ESP32. La aplicación, una vez el usuario acceda a la pantalla inicial y de manera transparente, será capaz de seleccionar el mejor punto de acceso y conectarse a él. En caso contrario, bien porque no lo ha encontrado o bien porque no ha conseguido conectarse, muestra un mensaje de error al usuario.

De manera nativa App Inventor no dispone de la funcionalidad de conexión con redes Wi-Fi pero, del mismo modo que las notificaciones, se ha utilizado la extensión TaifunWifi creada por Taifun (2021). La única consideración que debe tenerse en cuenta para usar esta extensión es que necesita disponer del sensor de localización del dispositivo. En este caso este componente sí es nativo de App Inventor y está disponible dentro de la sección de sensores.

Esta extensión tiene distintos métodos para operar con las redes Wi-Fi como comprobar el listado de redes Wi-Fi guardadas en el dispositivo, obtener un listado de redes Wi-Fi disponibles, señal de cada SSID, etc.

Aunque la operación de conexión contra el punto de acceso es transparente al usuario, hay una operación que no es posible ejecutarla de manera automática y es habilitar los permisos de acceso al sensor de localización del teléfono. En caso que los permisos no estén habilitados se muestra un mensaje informativo para que el usuario acepte expresamente estos permisos (figura 49).

***Figura 49.*** *Comprobación de permisos de acceso a la localización del dispositivo móvil.*




Fuente: App Inventor, 2021.

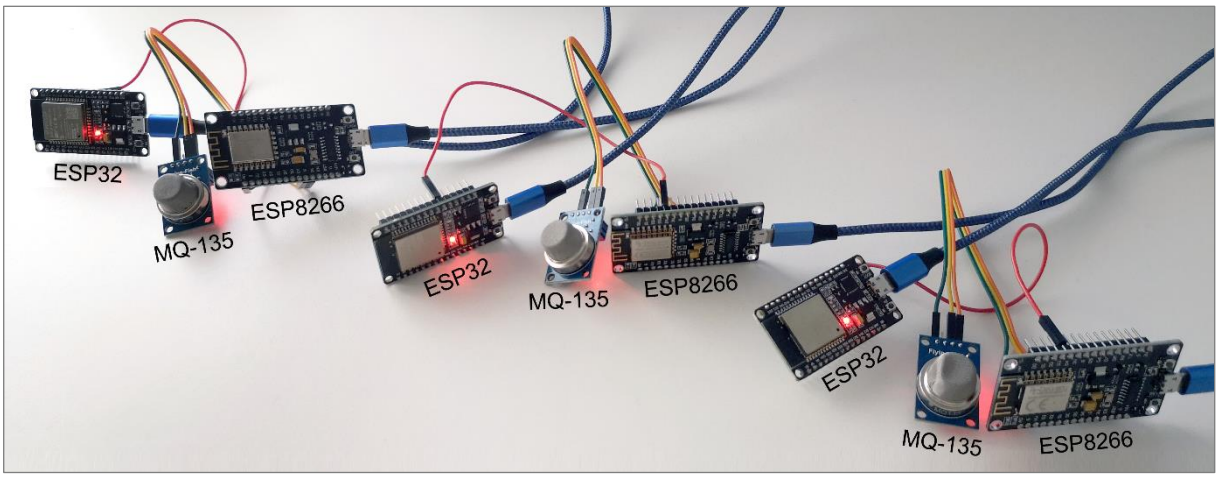
Tras habilitar los permisos de acceso a este sensor, se realiza la llamada al método AvailableSSIDs para obtener todos los SSIDs y recuperar solamente aquellos que empiecen por “ESP32-Sensor”. Este nombre es el que se ha configurado como nombre del punto de acceso creado con el microcontrolador ESP32.

Después de conectarse de manera satisfactoria al punto de acceso, con el método *LocalIP* de la extensión *TaifunWifi* obtenemos la IP del punto de acceso y se guarda dentro del namespace de valores de configuración de la aplicación. Esta operación se realiza cada vez que se accede a la aplicación para construir correctamente las peticiones HTTP de la pantalla de obtención de valores.

# 5. Evaluación de la solución

En este apartado se abordarán las pruebas necesarias para validar el correcto funcionamiento de la solución planteada. Para poder realizar dichas pruebas correctamente, se han utilizado los componentes reales. Tal y como se definió la arquitectura en el apartado anterior, la solución está compuesta por tres dispositivos ESP32, tres dispositivos ESP8266 y tres sensores MQ-135. Con el fin de monitorizar las entradas y las salidas en cada dispositivo, se han conectado todos ellos a los puertos USB del ordenador de modo que, utilizando el software Arduino CC, se podrá visualizar las trazas escritas por el puerto serie de cada uno de estos dispositivos.

***Figura 50.*** *Implementación de la arquitectura.*




Fuente: Elaboración propia, 2021.

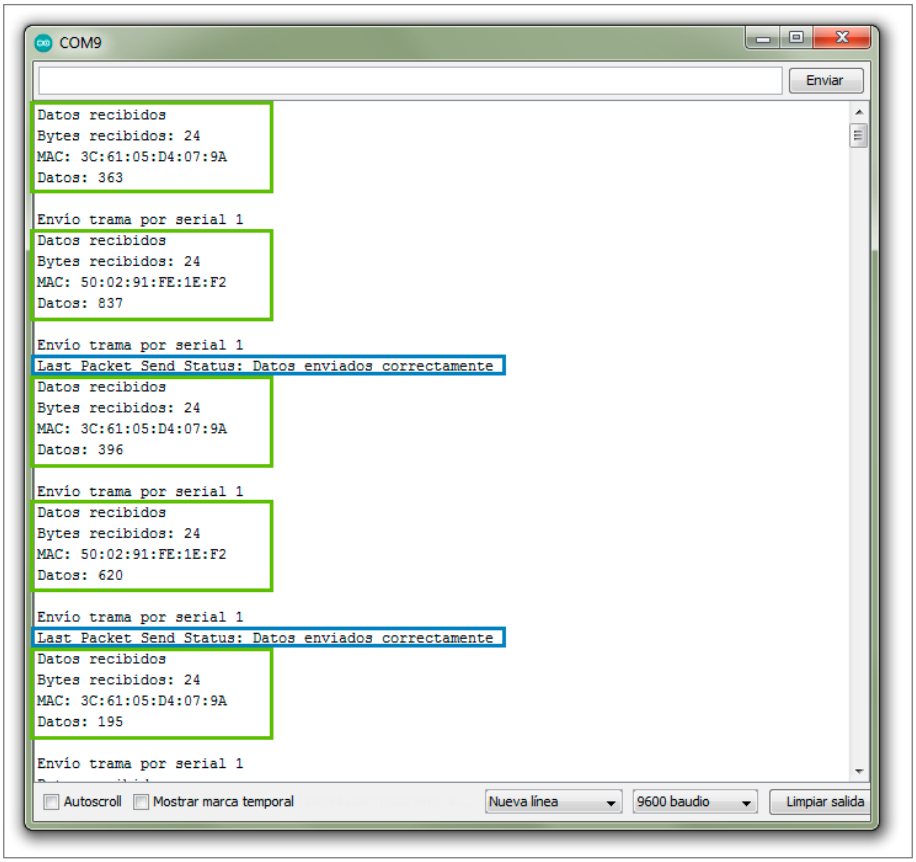
## 5.1. Pruebas de los elementos IoT

### 5.1.1. Pruebas de envío de valores por ESP-NOW

Por simplicidad en las pruebas, el valor del sensor es un valor aleatorio entre 0 y 1023 generado con la función matemática random(). Una vez validada las pruebas que se envían y reciben datos correctamente, se procedería a conectar el sensor y habilitar la lectura de dichos datos y el cálculo de la concentración.

La primera prueba realizada ha sido conectar los tres dispositivos ESP8266 y monitorizar el puerto serie. Este microcontrolador traza cuando ha realizado el envío indicando si ha sido correcto o incorrecto y traza cuando recibe un paquete, en ese caso añade la información recibida como la dirección MAC del dispositivo que la envía y el valor del sensor (que en el caso de estas pruebas es ficticio).

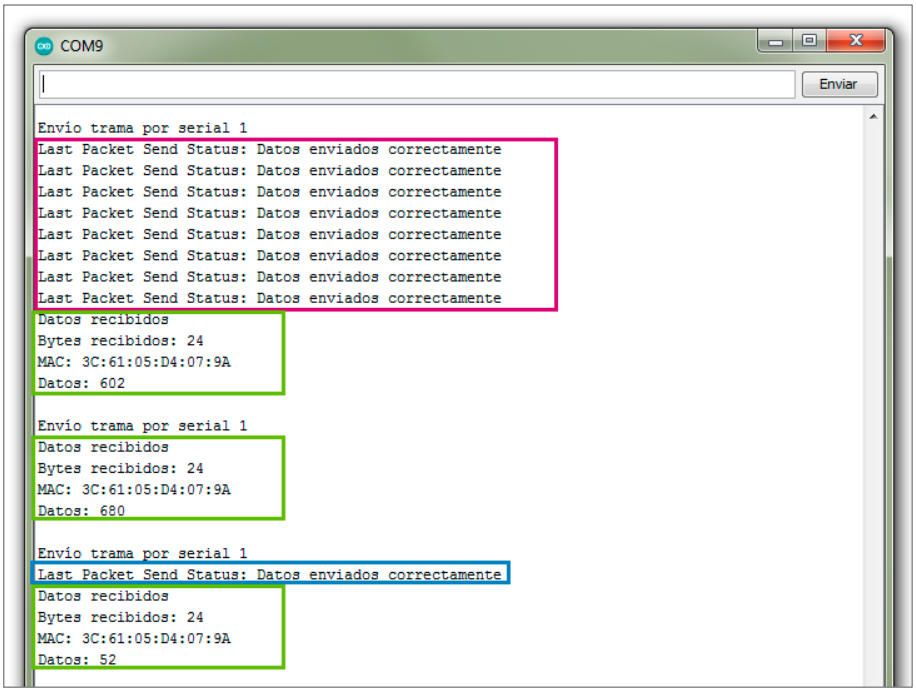
***Figura 51.*** *Prueba de conectividad a través de ESP-NOW.*




Fuente: Arduino CC, 2021.

Como se puede apreciar en la figura 51, tras habilitar el puerto COM9 en el Arduino CC, se trazan los datos recibidos (remarcado en verde) y también se trazan todos los envíos realizados por el propio dispositivo (remarcado en azul). Tras esta prueba, el siguiente paso es desconectar dos dispositivos y comprobar que el único dispositivo conectado sigue trazando correctamente y enviando datos (aunque no haya ningún dispositivo receptor).

***Figura 52.*** *Prueba de desconexión y conexión de los dispositivos.*




Fuente: Arduino CC, 2021.

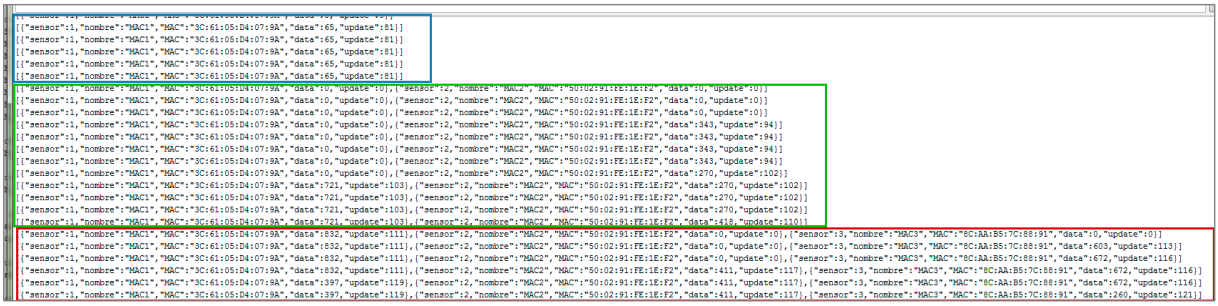
En la figura 52 muestra la evidencia dónde por un período corto de tiempo el dispositivo no recibió ninguna petición (remarcado en magenta) y tras la conexión de otro dispositivo ya vuelve a recibir las peticiones correctamente. En esta figura solamente se visualizan los datos de un único dispositivo porque solamente hay dos dispositivos conectados, uno recibiendo y otro enviando. En el momento que se conecta el tercer dispositivo las trazas vuelven a mostrar un comportamiento similar al mostrado en la figura 51.

### 5.1.2. Pruebas de envío de valores por puerto serie

Otro punto crítico en este proyecto que requiere de evaluación y pruebas es la recepción de datos por el puerto serie. Como comentamos en apartados anteriores, la lectura por el puerto serie se realiza byte a byte y podría darse el caso de recibir tramas corruptas. Para ello se han habilitado los flags de inicio, fin de trama y control de longitud.

Para la ejecución de estas pruebas se han conectado los tres ESP8266 y un único ESP32, todos ellos conectados por USB al ordenador. Además uno de los ESP8266 se ha conectado físicamente al puerto serie del ESP32. También, como para esta prueba todavía no está implementada la funcionalidad de administración de dispositivos por la aplicación, se han introducido manualmente las direcciones MAC en el código del propio ESP32.

***Figura 53.*** *Prueba de recepción de datos por el puerto serie en el ESP32.*

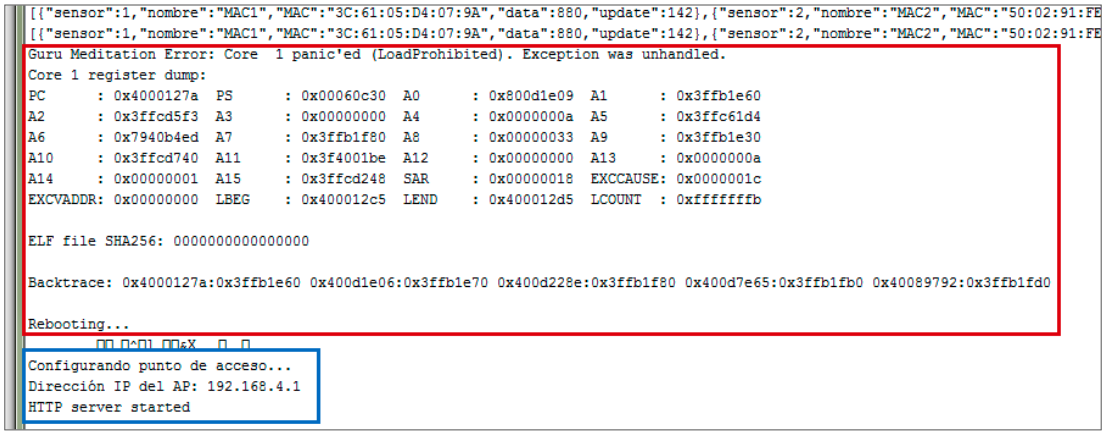



Fuente: Arduino CC, 2021.

Como se puede apreciar, en la figura 53 se han diferenciado tres secciones, azul, verde y roja. Inicialmente el servidor tenía establecido únicamente una dirección MAC por lo que aunque por el puerto serie recibiese información relativa a otras direcciones MAC solamente almacenaba la dirección MAC guardada. El siguiente paso, la sección verde, se ha establecido otro dispositivo, y como se puede apreciar el JSON ha duplicado su longitud y tiene dos dispositivos asociados. Finalmente, la sección roja, ya se contempla todo el JSON que el microcontrolador tiene guardado en su memoria.

Durante las pruebas cabe destacar que ha habido momentos donde el envío de bytes no funcionaba correctamente y aunque se tenía el control de la trama, el ESP32 se reiniciaba automáticamente. Tras varios días de pruebas con errores aleatorios y buscando en foros y comunidades, se llegó a la conclusión que el problema estaba derivado por una fluctuación indeseada de la tensión recibida en el puerto USB del dispositivo. Esto se pudo corroborar conectando los dispositivos a un banco de alimentación externa dónde la tensión siempre es constante y no se obtuvieron más errores durante una ejecución ininterrumpida durante horas.

***Figura 54.*** *Evidencia del error y reinicio automático del ESP32.*




Fuente: Arduino CC, 2021.

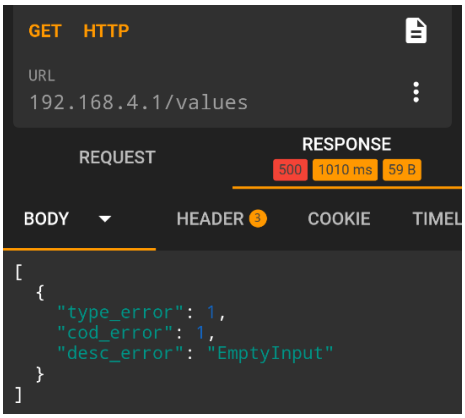
### 5.1.3. Pruebas del servidor web

La última prueba antes de empezar con las validaciones en la aplicación es comprobar que el servidor web es capaz de recibir peticiones y devolver las respuestas de manera adecuada. Para estas pruebas se ha utilizado la aplicación para terminales Android llamada Restler (2021). Esta aplicación permite elaborar peticiones HTTP con cualquier método (GET, POST, PUT,…) añadiendo parámetros en el propio cuerpo o por query string en la URL.

La primera prueba realizada es intentar provocar un error controlado en la recepción de valores. Para ello, sin tener guardado ningún dispositivo en memoria del ESP32, se ha lanzado la petición HTTP */values*. En el momento que el servidor web ha recibido la respuesta y ha intentado guardar el string en formato JSON, el método *deserializeJson* devuelve un error controlado con código 1.

Para dar por válida la prueba, esta petición debe devolver un código de respuesta HTTP 500 y en el cuerpo el JSON con el contenido del tipo de error, código de error y descripción del error. En la figura 55 se puede comprobar que tanto la salida como el código HTTP devuelto es el esperado.

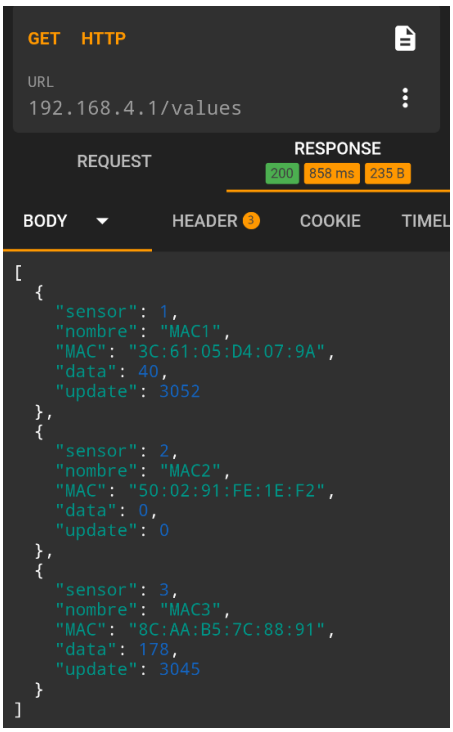
***Figura 55.*** *Error controlado cuando no hay dispositivos asociados.*




Fuente: Restler, 2021.

La siguiente prueba para la evaluación del correcto funcionamiento del servidor web es la ejecución de la petición HTTP con método POST para inicializar los dispositivos en memoria del servidor y posteriormente repetir la petición HTTP */values* y comprobar que ya no devuelve ningún error y se obtiene los valores y las direcciones MAC de los distintos dispositivos.

***Figura 56.*** *Ejecución correcta de obtención de valores con Restler.*




Fuente: Restler, 2021.

En figura 56 se puede comprobar que la prueba de validación es correcta dado que el código de respuesta HTTP es un 200 (petición correcta), el JSON recibido en el cuerpo de la petición tiene un formato correcto y los datos recibidos son coherentes.

## 5.2. Pruebas de la aplicación móvil

La evaluación de la aplicación móvil creada con App Inventor es el último paso para completar todas las pruebas de evaluación de la solución implementada para este proyecto. Durante la fase de diseño de pantalla, navegaciones, disposición de botones, gráficos, etc., App Inventor te ofrece la posibilidad de realizar un debug conectando el dispositivo por USB al ordenador y visualizar en tiempo real cómo quedan los elementos dispuestos en la pantalla de tu dispositivo móvil.

Esta funcionalidad de App Inventor es limitada, no te permite hacer todas las pruebas de algunas de las funcionalidades. Un ejemplo ha sido la realización de pruebas del componente de escaneo y conexiones a redes Wi-Fi, App Inventor te indica que es necesaria la compilación de la aplicación y posterior instalación en el dispositivo.

### 5.2.1. Pruebas de administración de dispositivos

Una de las pantallas principales de la aplicación es la de administración de dispositivos asociados. Las pruebas que se han realizado para validar correctamente la funcionalidad es que pueda añadir, modificar y borrar dispositivos. También deberá responder correctamente a las inconsistencias que se pueden producir durante la introducción de datos por parte del usuario como son caracteres erróneos, por ejemplo en la dirección MAC, comprobar longitud de los campos y que no haya campos vacíos.

En la figura 57 se puede comprobar cómo la aplicación muestra por pantalla textos indicando que hay errores en los campos de entrada y los colorea de color amarillo indicando donde está el error (inicialmente todos son de color blanco). Además, en la figura se puede

comprobar cómo ya existe un dispositivo asociado que se ha etiquetado como “Cocina” y se muestra en el desplegable.

***Figura 57.*** *Ejemplo de control de errores en campos de entrada de la aplicación.*


Fuente: Elaboración propia, 2021.

### 5.2.2. Pruebas de visualización de datos

Después de comprobar que la administración de los dispositivos funciona correctamente, el siguiente punto a evaluar sobre la aplicación es la pantalla de visualización de datos. En esta pantalla se podrá comprobar que la aplicación recupera los dispositivos almacenados en la base de datos que dispone la aplicación, es capaz de construir la petición HTTP a través de POST para inicializar los dispositivos en el servidor web, construir la petición HTTP a través de GET para obtener los valores y finalmente representarlos gráficamente.

El umbral configurado durante las pruebas ha sido de 150 ppm que es considerado un valor peligroso para la salud en cuanto a calidad del aire. En la figura 58 se puede apreciar como la aplicación muestra un mensaje indicando que la calidad del aire es mala o peligrosa y

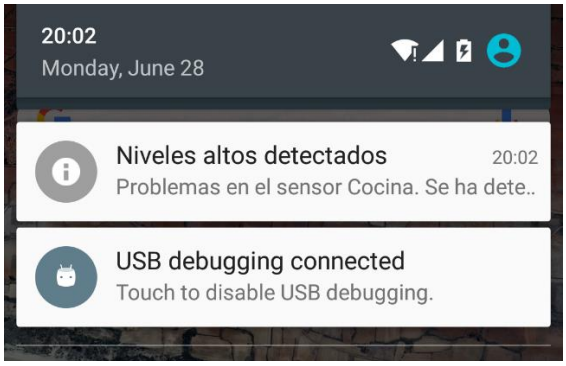
además muestra en un gráfico circular el valor. Este gráfico toma el valor máximo que puede capturar el sensor para el nivel de CO que es 200 ppm.

***Figura 58.*** *Ejemplo de visualización de datos de los sensores.*


Fuente: Elaboración propia, 2021.

También, se ha podido validar que la aplicación muestra una notificación indicando qué sensor ha superado el valor del umbral definido, en este caso el sensor de la Cocina (figura 59). Si el usuario pulsa en la notificación accederá a la pantalla de visualización de datos.

***Figura 59.*** *Ejemplo de visualización de la notificación.*




Fuente: Elaboración propia, 2021.

# 6. Conclusiones y Trabajos Futuros

## 6.1. Conclusiones

Con el desarrollo de este TFM se ha podido llevar a cabo la implementación de un prototipo de aplicación móvil capaz de capturar la calidad del aire recogida por un sensor e indicar si estos valores son normales o peligrosos para la salud. Además de la obtención de los valores, esta aplicación es capaz de generar una alerta cuando estos sensores capturen valores que estén por encima de un umbral definido por el usuario.

Este proyecto partió de un problema real, cotidiano, como es la intoxicación por inhalación involuntaria de gases. Aunque probablemente esta propuesta podría valer para entornos industriales, la solución está pensada para uso en hogares debido a que es el lugar dónde se producían la mayor parte de las intoxicaciones. Por otra parte, de cara a definir la solución de arquitectura, se ha tenido en cuenta el rango de edad sobre el que va dirigida esta solución, de modo que, al tratarse de personas con avanzada edad, la solución propuesta tuvo en cuenta que debía funcionar sin conectividad a internet ni dispositivos adicionales como routers, hubs, etc.

Para lograr el objetivo principal, primeramente se plantearon unos objetivos más específicos, después se implementaron, se evaluaron individualmente y finalmente se hizo una evaluación conjunta. Los objetivos específicos definidos y el resultado han sido:

- Comunicación con el sensor MQ-135: Se implementó el código necesario para obtener la información. El resultado de las pruebas fueron satisfactorias.
- Comunicación entre microcontroladores ESP8266: Se desarrolló el código necesario para que los distintos microcontroladores se comunicasen. Además, se probó la conexión automática tras un reinicio inesperado. El resultado de las pruebas fueron satisfactorias.
- Comunicación entre ESP32 y ESP8266: Se definió un protocolo de comunicación para el envío de datos por el puerto serie entre ambos dispositivos. El resultado de las pruebas fueron satisfactorias.

- Creación de un servidor web: Utilizando el microcontrolador ESP32 se consiguió crear un servidor web que fuese capaz de enviar información a través de peticiones HTTP. El resultado de las pruebas fueron satisfactorias.
- Creación de la APP: Aprovechando las facilidades que presenta App Inventor para desarrollo de aplicaciones para dispositivos Android, se elaboró un prototipo de aplicación capaz de solicitar información y mostrar los datos. El resultado de las pruebas fueron satisfactorias.

Además de conseguir los objetivos planteados, abordar este trabajo fin de máster ha servido para consolidar los conocimientos adquiridos durante el transcurso de este máster con asignaturas de ingeniería de software, lenguajes de desarrollo web, administración de servidores, IoT, etc. Del mismo modo, también ha servido para contrastar estas ideas y conceptos teóricos con resultados reales y los errores que conlleva.

Un ejemplo de ello ha sido la implementación de la comunicación entre los dos microcontroladores por el puerto serie. Este objetivo específico tal vez ha sido el más complicado de llevar a cabo por los posibles errores aleatorios de los dispositivos cuando enviaban y recibían la información por el puerto serie. Este factor inicialmente no se tuvo en cuenta en el diseño pero posteriormente se implementaron mejoras para controlar estos errores durante la comunicación.

## 6.2. Trabajos Futuros

Como líneas de trabajo futuras, al tratarse de un prototipo, el margen de mejora y evolución que presenta es bastante amplio. Algunas de las propuestas de trabajo futuras podrían ser:

1- Mejora de la usabilidad y accesibilidad de la aplicación: Esta línea de trabajo podría adaptar las pantallas, botones, navegaciones, etc. para que sean más accesibles y adaptadas al público objetivo con el fin de mejorar la experiencia de uso.

2- Utilización de servicios Cloud para almacenar la información: Adaptar la aplicación para que si el terminal sobre el que está ejecutándose dicha aplicación tiene conectividad a internet, enviar los datos recopilados a un servidor en la nube y que estos datos sean consultables de manera remota o desde otro dispositivo.
3- Configuración y calibración de distintos tipos de gases: Esta aplicación está implementada para interpretar un tipo de gas. Añadir más funcionalidad a la aplicación para administrar qué cada sensor pueda medir un gas específico y además poder utilizar valores de referencia de la ciudad para calibrarse de manera automática conectándose a servicios web/APIs disponibles.
4- Mejora de la seguridad en la comunicación: Aunque los datos que se manejan no son datos sensibles, podría darse el caso que un usuario malintencionado pueda interferir en las comunicaciones entre los distintos microcontroladores. Para ello se puede indicar en cada microcontrolador qué lista de direcciones MAC acepta para el envío y recepción de datos.
5- Optimización del uso de energía: Los microcontroladores ESP32 y ESP8266 tienen la capacidad de quedarse en modo “Deep Sleep”. Este modo de actividad ayuda a ahorrar batería y utilizando el mecanismo de interrupciones hardware y software pueden despertar y empezar recopilar datos de manera instantánea.
6- Uso del Bluetooth® para la comunicación con el servidor: El microcontrolador ESP32 puede trabajar con el protocolo BLE (Bluetooth® Low Energy) para comunicarse con la aplicación. Este protocolo de comunicación consume menos recursos energéticos que la comunicación Wi-Fi y por tanto se podría optimizar el uso de energía.

Para finalizar, haber desarrollado una solución a un problema real que sea de bajo coste, de código abierto y no depender de soluciones cerradas o propietarias, ha sido muy motivador y ha supuesto una superación a nivel personal. Poder transformar esta idea inicial en un prototipo funcional, que sirva a otras personas como base para poder seguir con el desarrollo de mejoras y que el proyecto forme parte de todos es altamente gratificante.

# Referencias bibliográficas

App Inventor. (14 de junio de 2021). *MIT App Inventor*. Obtenido de http://appinventor.mit.edu/

Arduino CC. (27 de junio de 2021). *Arduino*. Obtenido de https://www.arduino.cc

Artero, Ó. T. (2013). *ARDUINO. Curso práctico de formación.* RC Libros.

ASPgems. (20 de junio de 2021). *ASPgems*. Obtenido de https://aspgems.com/metodologia-de-desarrollo-de-software-iii-modelo-en-espiral/

AZ-Delivery. (24 de junio de 2021). Obtenido de AZ-Delivery: https://www.az-delivery.de/

Bartolomé Navarro, M. T., Amores Valenciano, P., Cuesta Vizcaíno, E., & Gallego Giménez, N. (2010). Intoxicación por Monóxido de Carbono: Una patología poco valorada en Urgencias. *Revista Clínica de Medicina de Familia*, 220-222.

Buchelli Ramirez, H., Fernández Álvarez, R., Rubinos Cuadrado, G., Martínez González, C., Rodríguez Jerez, F., & Casan Clara, P. (2014). Niveles elevados de carboxihemoglobina: fuentes de exposición a monóxido de carbono. *Archivos de bronconeumología*, 465–468.

Cameron, N. (2021). ESP-NOW and LoRa communication. En *Electronics Projects with the ESP8266 and ESP32* (págs. 365-397). Apress, Berkeley, CA.

Circuits4you. (25 de junio de 2021). *Circuits4you*. Obtenido de https://circuits4you.com/2018/12/31/esp32-devkit-esp32-wroom-gpio-pinout

Fleta Zaragozano, J., Fons Estupiñá, C., Arnauda Espatolero, P., Ferrer Dufol, A., & Olivares López, J. L. (2005). Intoxicación por monóxido de carbono. *Anales de Pediatría*, Vol. 62, No. 6, pp. 587-590.

Fouladi, B., & Ghanoun, S. (2013). Security evaluation of the Z-Wave wireless protocol. *Black hat USA*, 24, 1-2.

Gislason, D. (2008). *Zigbee wireless networking.* Newnes.

Heiman. (14 de mayo de 2021). *Heimantech*. Obtenido de http://www.heimantech.com/product/?type=detail&id=47

INE. (2021). *Instituto Nacional de Estadística*. Obtenido de https://www.ine.es/

Maayan, G. D. (13 de enero de 2020). *The IoT Rundown For 2020: Stats, Risks, and Solutions*. Obtenido de Security Today: https://securitytoday.com/Articles/2020/01/13/The-IoT-Rundown-for-2020.aspx

Macías Robles, M. D., Fernández Carreira, J. M., García Suárez, I., Fernández Diéguez, O., & Redondo Torres, G. (2009). Evolución epidemiológica de las intoxicaciones agudas por gases tóxicos atendidas durante el periodo de 2004 a 2007 en urgencias de un hospital comarcal. *Emergencias: Revista de la Sociedad Española de Medicina de Urgencias y Emergencias*, 350-353.

Maier, A., Sharp, A., & Vagapov, Y. (2017). Comparative analysis and practical implementation of the ESP32 microcontroller module for the internet of things. *2017 Internet Technologies and Applications (ITA)*, 143-148.

Restler. (27 de junio de 2021). *Repsositorio GitHub*. Obtenido de https://github.com/tiagohm/restler

Sheng, Z., Mahapatra, C., Zhu, C., & Leung, V. (2015). Recent advances in industrial wireless sensor networks toward efficient management in IoT. *IEEE access*, 3, 622-637.

Taifun. (26 de junio de 2021). *Pura Vida Apps*. Obtenido de https://puravidaapps.com/notification.php

Thom, S. R. (2002). Hyperbaric-oxygen therapy for acute carbon monoxide poisoning. *New England Journal of Medicine*, 1105-1106.

Wolber, D. (2011). App inventor and real-world motivation. *Proceedings of the 42nd ACM technical symposium on Computer science education*, 601-606.

Wolber, D., Hal, A., Spertus, E., & Looney, L. (2011). *App Inventor.* O'Reilly Media, Inc.