Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Sistema distribuido para determinar el nivel de CO2 en el aire

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Pintoiu, David Gabriel

Tutor: Blanc Clavero, Sara

Curso 2019/2020

Resumen

En los últimos años, la sociedad se ha visto afectada por la contaminación del medio ambiente a raíz de la acción de las propias personas, convirtiéndose así en una de las temáticas más polémicas adquiriendo un gran nivel de popularidad. Por ello, junto con la evolución de la tecnología en el ámbito de la domótica, se ofrece un servicio mediante el cual se puede aclarar un poco más la incógnita acerca de cómo es el entorno que nos rodea. El sistema en cuestión ofrece la posibilidad de monitorizar el nivel de CO_2 en el aire, así como la temperatura y la humedad de este. La distribución se basa en un sensor y una placa *Arduino* que capturan los datos y los envía a un servicio de base de datos en la nube ofrecida por *Firebase*. De manera paralela, una aplicación móvil desarrollada en *Flutter*, obtiene los datos y los representa a través de una interfaz de usuario pensada en ofrecer una visualización clara y rápida del estado actual del entorno donde se encuentra el sensor. Además, tanto la placa como la aplicación permiten manejar un actuador de forma automática o manual.

Palabras clave: medio ambiente, contaminación, domótica, aire, CO_2, *Arduino*, *Flutter*, *Firebase*.

Resum

En els últims anys, la societat s'ha vist afectada per la contaminació del medi ambient arran de l'acció de les mateixes persones, s'ha convertit així en una de les temàtiques més polèmiques adquirint un gran nivell de popularitat. Per això, juntament amb l'evolució de la tecnologia en l'àmbit de la domòtica, s'ofereix un servei mitjançant el qual es podrà aclarir una mica més la incògnita sobre com és l'entorn que ens envolta. El sistema en qüestió ofereix la possibilitat de monitorar el nivell de CO_2 en l'aire, així com la temperatura i la humitat d'aquest. La distribució es basa en un sensor i una placa *Arduino* que capturen les dades i els envia a un servei de base de dades en el núvol oferit per *Firebase*. De manera paral·lela, una aplicació mòbil desenvolupada en *Flutter*, obté les dades i els representa a través d'una interfície d'usuari pensada a oferir una visualització clara i ràpida de l'estat actual de l'entorn on es troba el sensor. A més, tant la placa com l'aplicació permeten manejar un actuador de manera automàtica o manual.

Paraules clau: medi ambient, contaminació, domòtica, aire, CO_2, *Arduino*, *Flutter*, *Firebase*.

Abstract

In recent years, society has been affected by pollution of the environment because of the actions of people themselves, becoming one of the most controversial topics, acquiring a high level of popularity. For this reason, together with the evolution of technology in the field of home automation, a service is offered with the purpose of clarifying a little more the unknown about what the environment surrounds us is like. The system in question offers the possibility of monitoring the level of CO2 in the air, as well as its temperature and humidity. The distribution is based on a sensor and an *Arduino* board that capture the data and send it to a *Cloud* database service offered by *Firebase*. In parallel, a mobile application developed in *Flutter*, obtains the data and represents it through a user interface designed to offer a clear and fast visualization of the current state of the environment where the sensor is located. In addition, both the platform and the application allow you to operate an actuator automatically or manually.

Key words: environment, pollution, home automation, air, CO_2, *Arduino*, *Flutter*, *Firebase*.

Tabla de contenidos

Tabla de figuras

1. Introducción

Hoy en día, la contaminación atmosférica es uno de los principales problemas que preocupan gravemente a la humanidad. La concentración de CO_2 en la atmósfera es uno de los gases producidos que más inquietan. Además, a causa de la contaminación, el medio ambiente ha sufrido potentes y nefastos cambios irreversibles.

Todo esto ha derivado en políticas y medidas nacionales en las que está previsto que los países de la UE y otros como China, Brasil o la India cumplan parte de sus objetivos de reducción de emisiones mediante inversiones en proyectos de prevención en materia medioambiental.

En los últimos años, la tecnología ha ido avanzando a pasos agigantados, estando muy presente en nuestras vidas. Los sistemas basados en el paradigma *IoT* son los principales causantes de este nuevo modelo de sociedad, acaparando desde los lugares más remotos hasta el rincón más pequeño de nuestros hogares.

En este nuevo escenario, donde los sensores y microcontroladores toman un papel muy relevante, la plataforma *Arduino* es uno de los protagonistas contando con un arsenal de placas reconocidas y utilizadas en todo el mundo.

Por otro parte, existen diferentes servicios *IoT* preparados y cualificados para dar soporte a la gran cantidad de microcontroladores que precisan una base de comunicación. Una de las plataformas más populares que ofrece dicho soporte es *Firebase*.

Nada de lo anterior tendría sentido sin un buen sistema para poder monitorizar la gran cantidad de datos producidos en este paradigma. Es por ello y por la determinación de los dispositivos móviles en la presente sociedad que *Flutter*, una plataforma de desarrollo ofrecida por *Google* al igual que *Firebase*, se convierte en un gran aliado para el propósito de desarrollar una aplicación móvil de calidad y multiplataforma que permita monitorizar todos los datos.

1.1. Motivaciones

La *Organización Mundial de la Salud*[1], señala que los niveles de polución del aire provocan diversas enfermedades, alteraciones y reducción de esperanza de vida en millones de personas de todo el mundo, aparte de causar impactos negativos en el medio ambiente. Es evidente que la mejor solución para ello es un mayor control de las actividades humanas y su influencia en la atmósfera.

Considero que conocer lo que está ocurriendo a nuestro alrededor es el primer paso para lograr un planeta mejor, por ello el problema global de la contaminación atmosférica ha sido una de las razones que me han impulsado a la realización del proyecto.

Por otro lado, los sistemas embebidos formados por microcontroladores y sensores es un área que nunca he pisado y me gustaría experimentar en el desarrollo de un servicio que incluya estos sistemas. Además, van directamente relacionados con la motivación anterior.

Otro de los motivos para llevar a cabo este proyecto, es el de realizar una aplicación para dispositivos móviles ya que es uno de mis pasatiempos favoritos.

1.2. Objetivos

Los objetivos planteados para el presente proyecto constan de los siguientes puntos.

- Desarrollar un sistema embebido formado por un microcontrolador y un sensor con la capacidad de analizar el medio ambiente en términos de calidad de aire.
- Configurar y desarrollar una plataforma *IoT* como base de comunicación entre el sistema embebido y una aplicación móvil.
- Desarrollar una aplicación móvil con la capacidad de monitorizar el sistema embebido y los datos que proporciona.

[1] https://www.who.int/es

1.3. Impacto esperado

En el presente trabajo, existe un único usuario principal que se ve afectado y este es cualquier persona u organización que disponga del sistema.

El impacto que se espera obtener incide en la conciencia del usuario acerca del entorno que le rodea a través de los datos obtenidos sobre la calidad de aire en términos de CO_2, temperatura y humedad.

Indirectamente, se espera una reacción por parte del usuario con el objetivo de mejorar la calidad de vida relacionándolo así con el objetivo de desarrollo número tres propuesto por la *UNESCO*[2], salud y bienestar.

Además, la originalidad de este proyecto pretende impactar en el objetivo número nueve, industria, innovación e infraestructura, también presentado por la *UNESCO*.

1.4. Metodología

El proyecto planteado sigue un modelo basado en el desarrollo incremental. Este modelo, se efectúa desarrollando pequeños bloques que requieren su correcto funcionamiento para avanzar con los siguientes.

1.5. Estructura

A continuación, se presentan los capítulos que componen el presente proyecto y se describen sus contenidos de manera general.

En el *capítulo 2 – Estado de la cuestión,* se introduce el contexto tecnológico acordado para este trabajo y las posibilidades que ofrece. Entre ellas se descubren varias plataformas para el desarrollo con microcontroladores, otras como bases de comunicación y multitud de F*rameworks* para el desarrollo de aplicaciones móviles. En este punto, también se comparan otros trabajos

[2] https://es.unesco.org/sdgs

parecidos al presente con el fin de demostrar la unicidad de este y finalmente se resumen las tecnologías escogidas para desarrollar.

En el *capítulo 3 – Análisis del problema*, se incluyen los requisitos para cumplir los tres objetivos propuestos. A través de tres diagramas se explican los requisitos de cada uno de los componentes del sistema, siendo estos el nodo formado por una placa y un sensor, una aplicación móvil y una plataforma *IoT* para el soporte de comunicación entre ambos. También se exponen la seguridad del sistema, la propiedad intelectual, la solución propuesta ante los requisitos establecidos, el plan de trabajo y el presupuesto.

En el *capítulo 4 – Diseño de la solución*, se explican tres puntos clave para llevar a cabo el diseño del sistema completo y los detalles de cada uno de los componentes por individual. Tanto en el nodo como en la aplicación móvil, se especifica la arquitectura que tienen y se detalla el funcionamiento interno de cada uno de ellos. Por otro lado, se especifica el diseño y la estructura de la base de datos.

En el *capítulo 5 – Desarrollo de la solución propuesta,* se explican las diferentes funcionalidades que ofrece el sistema completo. Este capítulo analiza los diferentes servicios obtenidos tras el diseño de cada uno de los componentes, indicando los resultados obtenidos por el sistema.

En el *capítulo 6 – Implantación,* se mencionan los requisitos para llevar a cabo la instalación y puesta en marcha del sistema.

En el *capítulo 7 – Pruebas* se explican las pruebas realizadas tanto en el nodo como en la aplicación móvil.

En el *capítulo 8 – Conclusiones* se exponen los resultados obtenidos en base a los objetivos marcados para el proyecto. También se mencionan los principales problemas encontrados y los conocimientos adquiridos tras completar el proyecto. Finalmente, se explica la relación entre los conceptos aplicados en el proyecto y los conocimientos adquiridos durante la carrera, así como las competencias transversales y específicas necesitadas para la realización del presente proyecto.

En el *capítulo 9 – Trabajos futuros* se explica el principal fleco de este proyecto y el planteamiento de una solución como trabajo futuro.

2. Estado de la cuestión

El *IoT* [1] [2] [3] [4] es un paradigma novedoso que está avanzando de manera eficaz en el ámbito de las telecomunicaciones inalámbricas modernas. La necesidad de mantener conjuntos de dispositivos conectados con el objetivo de intercambiar información fue uno de los principales motivos que dieron lugar a este nuevo modelo.

El patrón que define este paradigma está conformado por una red de dispositivos móviles identificables, capaces de interactuar entre ellos y con el entorno que les rodea utilizando tecnologías de comunicación especialmente diseñadas para este propósito [5] [6].

El presente proyecto se centra en explorar soluciones *IoT* completas. Estos sistemas están principalmente compuestos por tres componentes [7].

En primer lugar, se hallan los dispositivos programables formados por microcontroladores y sensores, que ofrecen la posibilidad de ejecutar la función de obtener datos medibles del entorno que los rodea y comunicarlos a través de la red. Existen varias familias de modelos en el mercado que presentan diferentes especialidades, pero todos se basan en cumplir las labores de adquirir datos, procesarlos y enviarlos. Las marcas más reconocidas en el mercado actual son *Arduino*[3] [8], *Raspberry*[4] y *Onion*[5].

Arduino presenta una gran variedad de placas, entre los cuales podemos encontrar los modelos *Arduino UNO* y *Arduino MKR Zero* pensadas para introducirse en el mundo de *Arduino* y proyectos más complejos respectivamente. También, existen dispositivos destinados a *IoT* como la serie *Arduino NANO* [9] o la reciente categoría *PRO* con nuevos dispositivos más potentes como *Arduino Portenta H7*.

Por otro lado, *Raspberry* cuenta con varias generaciones de dispositivos que van desde la generación *Raspberry Pi 1* con modelos como *Raspberry Pi 1 Model A+* hasta la serie 4 con placas como *Raspberry Pi 4 Model B*. Además, cuenta con la familia Zero con dos modelos, el *Raspberry Pi Zero* y *Raspberry Pi Zero W*.

[3] https://www.arduino.cc/pro

[4] https://www.raspberrypi.org/

[5] https://onion.io/

Otra de las marcas populares es *Onion*, que presenta dos modelos, el *Omega2* y el *Omega2S*. A pesar de solo tener dos placas, estas ofrecen unas especificaciones más que suficientes para el uso en el ámbito *IoT*.

Para que estas placas tengan sentido en los sistemas *IoT*, deben ir acompañados de dispositivos sensibles a estímulos ofrecidos por el entorno donde se hallan. Estos dispositivos se pueden encontrar en infinidad de aplicaciones, pues son los capaces de obtener cualquier propiedad cuantificable del medio, pero, este proyecto, se centra en los sensores cuya especificación sea el aire en términos de CO_2. Para este proyecto se han estudiado los modelos *MHZ-19B*[6] [10] y *SCD30*[7] [11]

En segundo lugar, todos los dispositivos de la red tienen una plataforma en común, conocida como plataforma *IoT* [12]. Esta, ofrece diferentes tipos de servicios para cubrir las amplias aplicaciones que pueden surgir en el campo del *IoT*, como por ejemplo almacenamiento en la nube, *Cloud Computing* [13] o *Cloud Messaging*.

Hay un mercado amplio que ofrece este tipo de plataformas entre las que se encuentran *Firebase*[8], *Azure Microsoft*[9], *Arduino Cloud*[10], *The Things Network*[11] y *AWS IoT Amazon*[12], entre otras.

En último lugar, y para completar la arquitectura *IoT*, se dispone de una herramienta capaz de controlar y monitorizar todos los dispositivos de la red. Una de las características de la arquitectura es ofrecer una gran accesibilidad, por tanto, desarrollar una aplicación para dispositivos móviles es una de las formas de cumplir este acometido.

[6] https://www.winsen-sensor.com/sensors/co2-sensor/mh-z19b.html

[7] https://www.sensirion.com/

[8] https://firebase.google.com/?hl=es

[9] https://azure.microsoft.com

[10] https://www.arduino.cc/en/IoT/HomePage

[11] https://www.thethingsnetwork.org/

[12] https://aws.amazon.com/es/iot/

Para ello, se hace uso de *Frameworks* [14] especializados en la creación de estas aplicaciones. Entre las más famosas podemos encontrar a *Android Studio*[13], *Flutter*[14], *Ionic*[15], *React Native*[16], *Xamarin*[17] y *Native* Script[18], entre otras.

2.1. Crítica al estado de la cuestión

Para confirmar la autenticidad y exclusividad de este proyecto se han seleccionado dos ejemplos que presentaban una arquitectura casi idéntica al presente trabajo. El primero de ellos [15], pertenece al curso académico 2016-2017, siendo más reciente que el segundo [16], que pertenece al curso 2014-2015.

En el primer ejemplo se utilizan cinco sensores y en el segundo dos, mientras que en este proyecto se hace uso de un único sensor.

El modelo de sensor que utiliza el primer trabajo para medir la temperatura y la humedad es *MP503 Air Quality Gas*, mientras que el segundo trabajo ha empleado el modelo *DHT11*. El sensor *SCD 30*, a diferencia de los ejemplos, ha sido el seleccionado para este proyecto.

El microcontrolador manejado en los tres proyectos forma parte de la plataforma *Arduino* diferenciándose únicamente el modelo seleccionado. *Arduino Yun* se encuentra en el primer ejemplo, *Arduino UNO R3* en el segundo ejemplo y en este trabajo se usa el modelo *Arduino NANO 33 IoT*.

Se ha escogido la plataforma *IoT* de *Firebase* a diferencia de los otros trabajos, que han decidido utilizar una placa Raspberry como servidor.

La representación de los datos obtenidos en el primer ejemplo se hace mediante una web, en cambio, en el segundo ejemplo, se representa mediante un programa realizado en *Phyton*. En este proyecto, se figura a través de una aplicación para dispositivos móviles.

[13] https://developer.android.com/studio/intro

[14] https://flutter.dev
[15] https://ionicframework.com/
[16] https://reactnative.dev/
[17] https://dotnet.microsoft.com/apps/xamarin
[18] https://nativescript.org/

Teniendo en cuenta estas diferencias, podemos afirmar con cierta seguridad que el proyecto planteado es único.

2.2. Propuesta

La finalidad de este proyecto es cumplir con los objetivos planteados en busca de una solución *IoT* completa. Para ello, se han estudiado las diferentes opciones propuestas, sin embargo, el reciente confinamiento causado por el *COVID-19* ha influido en la elección ya que la accesibilidad y disponibilidad de los productos se ha visto alterada.

Debido a que el proyecto no se centra en las prestaciones del sistema empotrado, se ha decidido escoger la placa *Arduino NANO 33 IoT*. Es un modelo comercial de una marca que, con sus nuevos productos está comenzando a posicionarse en el mercado del *IoT*.

En cuanto al sensor, el modelo *SCD30* ha sido el elegido por su accesibilidad además de ofrecer la posibilidad de analizar la temperatura y la humedad del ambiente.

La plataforma de comunicación y las herramientas para el desarrollo de la aplicación móvil se han seleccionado de manera conjunta para ofrecer la mayor compatibilidad posible. Estas son *Firebase* y *Flutter*, ambas ofrecidas por la compañía *Google*.

Trabajar con este conjunto ha facilitado la interoperabilidad entre ellas. También se ha producido dicha facilidad entre *Firebase* y la placa gracias al servicio *REST* que la plataforma ofrece. De esta manera, se ha podido focalizar más en el desarrollo del de cada uno de los componentes que en la unión entre ellos.

3. Análisis del problema

Ahora que ya se ha aclarado el contexto en el que se encuentra el proyecto, se procede a analizar el problema basándose en los objetivos propuestos.

Se pretende crear un servicio *IoT* a medida dotado con la capacidad de analizar la calidad del aire. Se centra, principalmente, en crear soluciones *IoT* completas. Dicho servicio, puede ser útil para diferentes entornos como hospitales, hoteles, colegios o a nivel particular.

Como primer objetivo a cumplir, se desarrolla un módulo de gestión de la información para que la placa, el sensor y el actuador, de ahora en adelante nodo, trabajen de manera conjunta en la obtención de datos y el envío de estos a una base de datos en la nube, así como la activación y desactivación del actuador.

Para la realización del segundo objetivo, se configura un servicio de base de datos en la nube y un servicio de mensajería sobre una plataforma *IoT*. Esta, ofrece soporte a los nodos que puedan existir y a la aplicación en términos de interconexión y comunicación.

Por último, se desarrolla una aplicación móvil con el propósito de poder visualizar tanto el estado de cada nodo en la red, como los datos capturados. Además, se puede visualizar y controlar el estado del actuador de cada uno de ellos.

3.1. Nodo

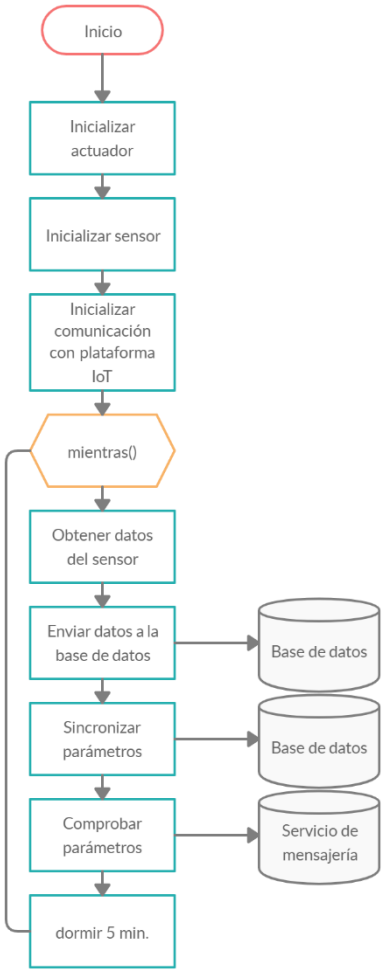


Figura 1. Diagrama de flujo del nodo.

El diagrama de flujo del sistema empotrado indica las funciones que éste debe realizar durante su ejecución. A continuación, se describen dichas funciones.

- Inicializar actuador: La placa se comunica con el actuador para inicializarlo y comprobar su correcto funcionamiento.
- Inicializar sensor: La placa se comunica con el sensor para inicializarlo y comprobar su correcto funcionamiento.
- Inicializar comunicación con plataforma *IoT*: La placa inicializa los parámetros para la posterior comunicación con la plataforma *IoT*.

Una vez realizadas las funciones de inicialización, la placa entra en un bucle infinito con un intervalo de ejecución de cinco minutos.

- Obtener datos del sensor: La placa se comunica con el sensor para obtener los datos capturados por el mismo.
- Enviar datos a la base de datos: La placa se comunica con la base de datos para enviar los datos obtenidos a través del sensor.
- Sincronizar parámetros: Se realiza una comunicación bidireccional entre la placa y la base de datos con el fin de actualizar el estado del sensor y sincronizarlo.
- Comprobar parámetros: Los datos obtenidos por el sensor son analizados contra los parámetros establecidos con el fin de enviar notificaciones y manejar el actuador según las necesidades.

3.2. Plataforma *IoT*

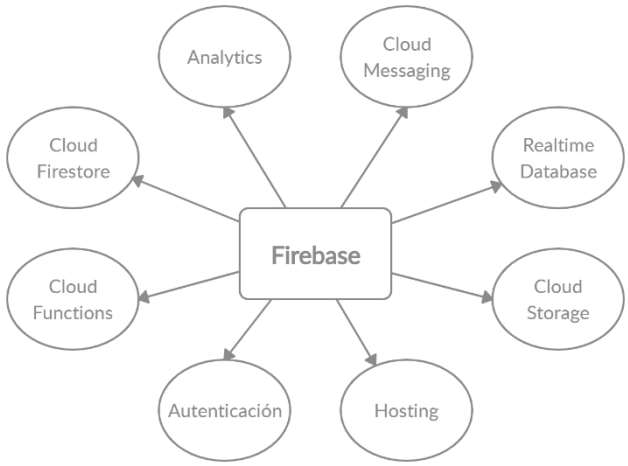


Figura 2. Plataforma IoT.

Como se ha comentado anteriormente, la plataforma *IoT* seleccionada es *Firebase*. Entre los diferentes servicios que ofrece, los únicos necesarios para llevar a cabo el desarrollo del proyecto son *Firebase Realtime Database*[x] y *Firebase Cloud Messaging*[x].

Firebase Realtime Database es un servicio de base de datos en la nube que presenta un modelo no relacional. Su arquitectura está fomentada por una estructura que sigue el estándar *JSON*. La gestión de datos en tiempo real, la asistencia sin conexión, gran accesibilidad y alta escalabilidad son algunas de sus funciones clave.

Firebase Cloud Messaging, es un servicio de mensajería que permite el envío y recepción de mensajes. Las principales características de este servicio son la posibilidad

de añadir datos dentro de los mensajes, el envío de estos de manera individual o colectiva y la seguridad y bajo consumo que ofrece su canal. Además, una de sus recientes novedades, es la incorporación de un servicio *REST* como método de envío, siendo este el que se ha utilizado en el sistema embebido.

3.3. Aplicación móvil

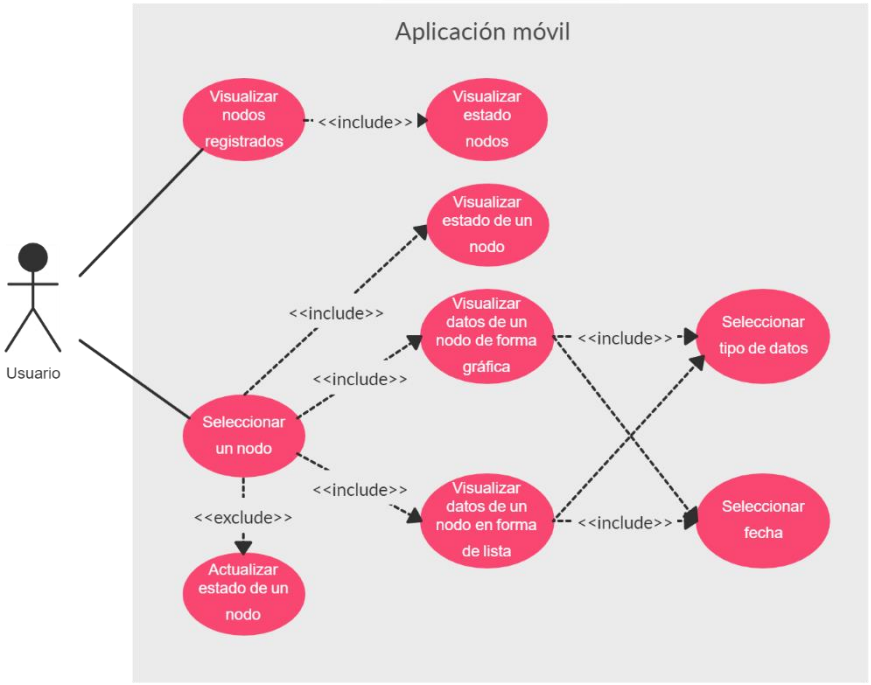


Figura 3. Casos de uso de la aplicación móvil.

Caso de uso	Visualizar nodos registrados.
Actores	Usuario
Resumen	El usuario puede visualizar los nodos registrados en la base de datos.
Precondiciones	Tener conexión a internet.
Postcondiciones	-
Incluye	Visualizar estado de los nodos.
Extiende	-
Flujo de eventos	
Actor	**Sistema**
1. El usuario entra en la aplicación.	2. Busca los sensores en la base de datos. 3. Muestra una lista de sensores en pantalla.

Caso de uso	Visualizar estado de los nodos.
Actores	Usuario
Resumen	El usuario puede visualizar el estado de cada nodo en tiempo real.
Precondiciones	Visualizar nodos registrados.
Postcondiciones	-
Incluye	-
Extiende	-
Flujo de eventos	
Actor	**Sistema**
	1. Crea y mantiene una conexión abierta con la base de datos y actualiza el estado de los nodos ante la recepción de eventos.

Caso de uso	Seleccionar un nodo.
Actores	Usuario
Resumen	El usuario puede seleccionar un nodo con el propósito de ver la información que ofrece ese nodo en concreto.
Precondiciones	Tener conexión a internet.
Postcondiciones	Se selecciona la lengüeta de gráfica.
Incluye	Visualizar datos en forma de gráfica. Visualizar datos en forma de lista. Visualizar estado de un nodo.
Extiende	Actualizar estado de un nodo.
Flujo de eventos	
Actor	**Sistema**
1. El usuario selecciona un nodo de la lista.	2. Muestra una pantalla con información del nodo.

Caso de uso	Visualizar datos de un nodo en forma de gráfica.
Actores	Usuario
Resumen	El usuario puede visualizar los datos ofrecidos por un nodo en particular en forma de gráfica.
Precondiciones	Tener conexión a internet.
Postcondiciones	Procesar los datos obtenidos.
Incluye	Seleccionar un tipo de datos (CO_2, Temperatura, Humedad). Seleccionar una fecha.
Extiende	-
Flujo de eventos	
Actor	**Sistema**
1. El usuario selecciona la lengüeta de gráfica.	2. Se buscan los datos en base al tipo y la fecha seleccionados. 3. Muestra una gráfica con los datos procesados.

Caso de uso	Visualizar datos de un nodo en forma de lista.
Actores	Usuario
Resumen	El usuario puede visualizar los datos ofrecidos por cada nodo en particular.
Precondiciones	Tener conexión a internet.
Postcondiciones	-
Incluye	Seleccionar un tipo de datos (CO_2, Temperatura, Humedad). Seleccionar una fecha.
Extiende	-
Flujo de eventos	
Actor	Sistema
1. El usuario selecciona la lengüeta de lista.	2. Se buscan los datos en base al tipo y la fecha seleccionados. 3. Muestra una lista con los datos.

Caso de uso	Visualizar estado de un nodo.
Actores	Usuario
Resumen	El usuario puede visualizar el estado del nodo seleccionado.
Precondiciones	Tener conexión a internet.
Postcondiciones	-
Incluye	-
Extiende	-
Flujo de eventos	
Actor	**Sistema**
	1. Crea y mantiene una conexión abierta con la base de datos con la referencia del nodo y actualiza su estado ante la recepción de eventos.

Caso de uso	Actualizar estado de un nodo.
Actores	Usuario
Resumen	El usuario puede modificar los parámetros de un nodo.
Precondiciones	Tener conexión a internet. Seleccionar la opción de configuración del nodo.
Postcondiciones	Se actualizan los datos en la base de datos.
Incluye	-
Extiende	-
Flujo de eventos	
Actor	**Sistema**
1. Selecciona el botón de configuración del nodo. 3. El usuario modifica los parámetros necesarios y confirma la actualización.	2. Se muestra una pantalla con la configuración modificable del nodo. 4. Se envía la configuración a la base de datos.

Caso de uso	Seleccionar tipo de datos (CO_2, Temperatura, Humedad).
Actores	Usuario
Resumen	El usuario puede seleccionar el tipo de datos a visualizar.
Precondiciones	Tener conexión a internet.
Postcondiciones	Se actualizan los datos visualizados.
Incluye	-
Extiende	-
Flujo de eventos	
Actor	**Sistema**
1. El usuario cambia el tipo de datos a visualizar.	2. Se buscan los datos del tipo seleccionado en la base de datos. 3. Se actualizan las listas y gráficas con los datos a mostrar.

Caso de uso	Seleccionar una fecha.
Actores	Usuario
Resumen	El usuario puede seleccionar la fecha a visualizar.
Precondiciones	Tener conexión a internet.
Postcondiciones	Se actualizan los datos visualizados.
Incluye	-
Extiende	-
Flujo de eventos	
Actor	**Sistema**
1. El usuario cambia la fecha a visualizar.	2. Se buscan los datos de la fecha seleccionada en la base de datos. 3. Se actualizan las listas y gráficas con los datos a mostrar.

3.4. Seguridad

La seguridad planteada para este proyecto se hereda de la plataforma *IoT*. Para el acceso a la base de datos y al servicio de mensajería a través del servicio *REST* se usan claves secretas proporcionadas por la plataforma, restringiendo el acceso a usuarios no autenticados. Además, las conexiones se efectúan a través del protocolo *SSL*, garantizando un gran nivel de seguridad.

Por otro lado, gracias a que tanto la plataforma *IoT* como el Framework usado para desarrollar la aplicación pertenecen a Google, no es necesario preocuparse por la seguridad de los datos en esta interconexión ya que la propia API ofrece un canal seguro.

3.5. Propiedad intelectual

En el producto final, se utilizan imágenes con licencias de libre uso con la única condición de nombrar a los autores, por tanto, a continuación, se ofrece una lista con dichos autores.

- "Icono hecho por Good Ware de www.flaticon.com " (**Anexo 1**)
- "Icono hecho por Freepik de www.flaticon.com " (**Anexo 2**)
- "Icono hecho por Xnimrodx de www.flaticon.com " (**Anexo 3**)
- "Icono hecho por Good Ware de www.flaticon.com " (**Anexo 4**)

3.6. Solución propuesta

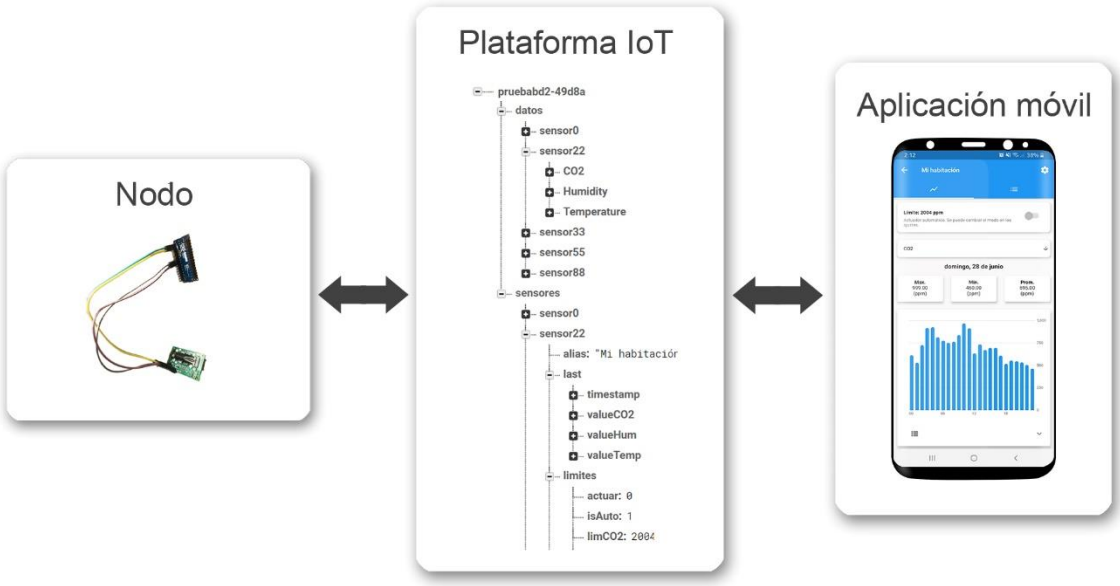


Figura 4. Sistema IoT completo.

Los requisitos de cada componente previamente establecidos dan lugar al prototipo de sistema distribuido que se busca en este proyecto. A la hora de unir las piezas, se obtiene:

- Un nodo encargado de obtener información y enviarla a una base de datos.

- Una plataforma *IoT* que incluye los servicios necesarios para mantener los datos y comunicar los dos extremos.
- Una aplicación móvil que permite visualizar y controlar los nodos.

Cada nodo está compuesto por la placa *Arduino NANO 33 IoT* y el sensor *SCD30* de *Sensirion*. Como actuador se usa el LED que tiene incorporada la placa con el fin de probar su correcto funcionamiento, aunque este puede cambiarse por un actuador real según las funciones que requiera ejecutar.

El sensor obtiene los niveles de CO_2, temperatura y humedad periódicamente y los envía a la placa utilizando una conexión bus de tipo I^2C cuando esta los requiera. Seguidamente, la placa prepara los datos obtenidos para enviarlos a la base de datos. Para ello, se usa el módulo *WiFi* que tiene incorporado, el cual establece una conexión *SSL* a través del servicio *REST* que ofrece la plataforma, registrando así, los datos obtenidos en la base de datos.

Además, la placa cumple la función de actualizar varios parámetros de su propio estado, entre ellos los límites de CO_2, temperatura y humedad. Una vez hecho esto, compara los datos obtenidos por el sensor con los parámetros. En caso de que alguno se vea superado, se envía una notificación a la aplicación móvil a través del servicio de mensajería, utilizando los mismos detalles de conexión que para la base de datos.

Por otro lado, si el modo de funcionamiento es automático, la placa activa o desactiva el actuador en caso de que los límites se hayan rebasado o no respectivamente. De lo contrario, si el modo es manual, el usuario indica desde la aplicación si el actuador debe activarse o no.

Para que la comunicación con el servicio de mensajería a través del protocolo *SSL* pueda realizarse desde la placa, es necesario configurar el módulo *WiFi* para obtener los certificados. El procedimiento para conseguir el certificado es el siguiente:

1. Entrar en *Arduino IDE*.
2. Conectar la placa al *PC*.
3. Seleccionar menú “Herramientas”.
4. Seleccionar la opción “*WiFi 101 / WiFiNINA firmware update*”.
5. Seleccionar “*Open Updater sketch*”.
6. Subir el *sketch* a la placa y esperar a que se complete.
7. En el menú anterior, seleccionar el puerto del módulo *WiFi* que aparece en la lista.

8. Seleccionar “*Add domain*”
9. Añadir “fcm.googleapis.com:443” y aceptar.
10. Seleccionar “*Upload Certificates to WiFi module*”.

Una vez completado el proceso, ya se puede utilizar el servicio de mensajería desde la placa. Cabe mencionar que, para usar el servicio de base de datos, se utiliza una librearía que incluye los elementos necesarios para efectuar una conexión segura.

Tanto para el servicio de mensajería como el de base de datos, se necesitan las claves secretas de los servidores para la autenticación. Estas, se puede conseguir en los ajustes del proyecto de *Firebase* el cual se indica como crear a continuación.

Para poder usar la plataforma *IoT*, se debe crear un proyecto dentro de *Firebase* y posteriormente activar y configurar los servicios a utilizar.

Primero se accede a la consola de *Firebase*[19]. Seguidamente, se añade un nuevo proyecto en el que se solicita un nombre y la activación de *Google Analytics*, que se debe activar con la cuenta por defecto de *Firebase*.

Una vez dentro del proyecto, se accede al servicio de base de datos desde el menú situado a la izquierda. A continuación, se muestran dos opciones para crear la base de datos, *Cloud Firestore* y *Realtime Database*, de las que se elige la segunda empezando con el modo de bloqueo.

Ahora que el servicio de base de datos en la nube está activo, falta por activar el servicio de mensajería. Para cumplir la activación se requiere en primer lugar activar *Google Analytics*, cosa que se ha hecho al crear el proyecto y la segunda consiste en habilitar el servicio desde la consola de desarrolladores de *Google*[20]. Los servicios para habilitar dentro de la consola son *Cloud Messaging* y *Firebase Cloud Messaging API*. Ambos se pueden encontrar en el buscador que hay en la sección de biblioteca.

Finalmente, para que la aplicación se pueda comunicar con *Firebase*, es necesario incluirla en el proyecto creado anteriormente. Las indicaciones para realizar este proceso se encuentran documentadas en las guías de *Firebase*[21].

La aplicación móvil debe cumplir con las funciones de control y de visualización. Para ello, una primera pantalla es la encargada de mostrar una lista con los nodos que en algún momento se hayan registrado en la base de datos. Cada entrada de la lista

[19] https://console.firebase.google.com/
[20] https://console.developers.google.com/apis/library
[21] https://firebase.google.com/docs/flutter/setup?hl=es

contiene la representación del estado de un nodo incluyendo su nombre, si está activo o inactivo, los últimos valores y un color u otro indicando si estos sobrepasan los límites establecidos para el sensor.

Una segunda pantalla se abre tras seleccionar uno de los nodos. Está compuesta por dos vistas que se pueden elegir a través de un panel de pestañas. La primera vista contiene el estado del actuador y el modo de funcionamiento del nodo. También se halla la opción de elegir el tipo de datos a mostrar y un gráfico de barras que muestra dichos datos en un rango de 24 horas. El actuador se puede activar o desactivar a conveniencia del usuario en caso de que el modo de funcionamiento sea manual, de lo contrario, no se puede modificar.

3.7. Plan de trabajo

Proyecto	**TOTAL 360H**
Análisis	70H
RF y RFN (Placa + sensor)	20H
Aspectos para tener en cuenta en la identificación	5H
Identificación de elementos	15H
RF y RFN BBDD	10H
Aspectos para tener en cuenta en la identificación	3H
Identificación de elementos	7H
Sistema de información (Aplicación)	40H
Elicitación	20H
Alcance del proyecto	6H
Prototipo inicial	12H
Requisitos	10H
Diagrama de casos de uso	10H
Validación	10H
Diseño	120H
Sistema de información (Placa + sensor)	30H
Diagrama de clases	15H
Esquemas del modelo del sistema de información	15H
Modelo de BBDD	20H
Prototipo estructura del árbol	10H
Esquemas modelo de BBDD	10H
Sistema de información (Aplicación)	70H
Capa de presentación	38H
Diseño interfaz de usuario	10H
Prototipos de interfaz de usuario	18H
Esquemas del modelo de la capa de presentación	10H
Capa lógica	20H
Diagrama de clases	9H
Esquemas del modelo de la capa lógica	11H

Capa de datos	12H
Diagrama de clases	5H
Esquemas del modelo de la capa de datos	7H
Implementación	140H
Sistema de información (Placa + sensor)	60H
Desarrollo del módulo de gestión del sistema de información de la placa y el sensor	60H
Sistema de información (Aplicación)	80H
Desarrollo del módulo de gestión del sistema de información de la aplicación móvil	80H
Pruebas	30H
Sistema hardware y de información (Placa + sensor)	15H
Resultado pruebas de componentes	10H
Resultado pruebas de integración	5H
Sistema hardware y de información (Aplicación)	15H
Resultado pruebas de componentes	10H
Resultado pruebas de integración	5H

3.8. Presupuesto

Servicio	**Coste**
Sensor SCD30	51,75 €
Arduino* NANO 33 *IoT	16,00 €
Horas/hombre	360*n €
TOTAL	**67,75 + 360*n €**

4. Diseño de la solución

En el presente capítulo, se explica en detalle el diseño de todos los componentes de la arquitectura. Para ello, primero se ilustran tres conceptos clave que permiten acatar el diseño de cada uno de forma más clara.

El primer concepto trata sobre el modelo del estado del nodo.

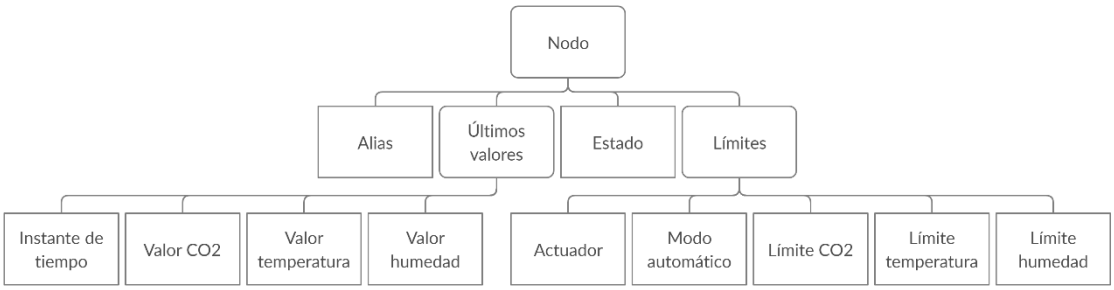


Figura 5. Modelo del estado del nodo.

La figura representa la composición del estado del nodo. Cada uno de los puntos terminales del esquema representa una propiedad. Estas se describen a continuación.

Propiedad	**Descripción**
Nodo	Define la referencia del nodo para la comunicación.
Estado	Define si el nodo se encuentra activo o inactivo.
Instante de tiempo	Define el instante de tiempo en el que se han registrado los últimos valores.
Valor CO_2	Define el último valor de CO_2 capturado por el nodo.
Valor temperatura	Define el último valor de temperatura capturado por el nodo.
Valor humedad	Define el último valor de humedad capturado por el nodo.
Actuador	Define si el actuador está encendido o apagado.
Modo automático	Define si el modo de funcionamiento es automático o manual. En el modo automático, el propio nodo controla el actuador. En el modo manual, el usuario controla el actuador a través de la aplicación.
Límite CO_2	Define el límite superior para el nivel de CO_2. Cuando dicho límite se ve superado, el nodo envía una notificación y pone en marcha el actuador si se encuentra en modo automático.
Límite temperatura	Define el límite superior para el nivel de temperatura. Cuando dicho límite se ve superado, el nodo envía una notificación y pone en marcha el actuador si se encuentra en modo automático.
Límite humedad	Define el límite superior para el nivel de humedad. Cuando dicho límite se ve superado, el nodo envía una notificación y pone en marcha el actuador si se encuentra en modo automático.

El segundo concepto clave es el modelo de los datos obtenidos por el nodo.

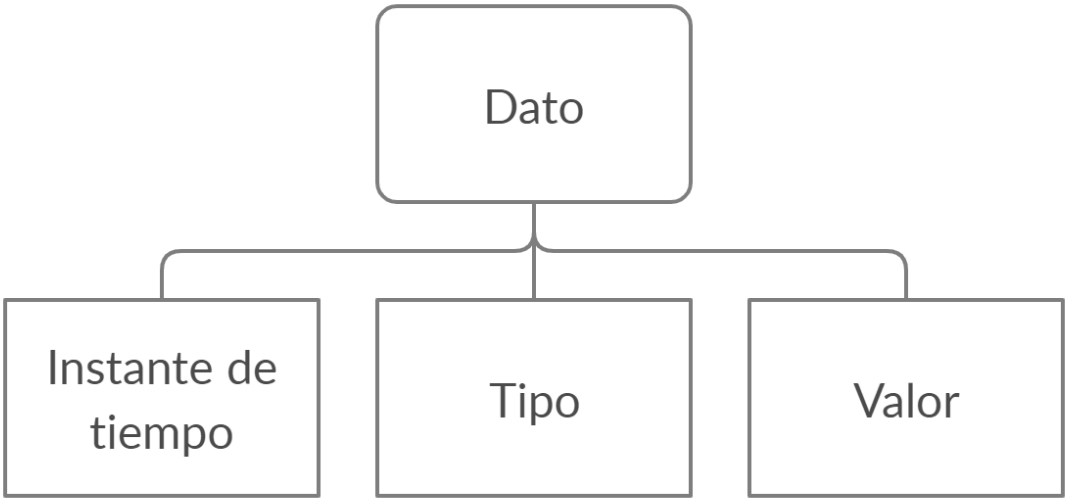


Figura 6. Modelo de dato.

A continuación, se describen sus propiedades.

Propiedad	**Descripción**
Instante de tiempo	Define el instante de tiempo en el que el dato se ha registrado en la base de datos.
Tipo	Define el tipo de dato capturado (CO_2, temperatura, humedad).
Valor	Define el valor capturado.

El tercer y último concepto clave consiste en definir las relaciones de lectura y escritura entre el nodo y la aplicación, para el estado del nodo.

Propiedad	**Nodo**		**Aplicación**	
	Lectura	Escritura	Lectura	Escritura
Alias	-	Una única vez al registrar el nodo en la base de datos	Cada vez que se actualiza el nodo en la base de datos.	Cada vez que el usuario lo solicite.
Estado	-	Cada iteración del bucle principal	Cada vez que se actualiza el nodo en la base de datos.	Cuando la aplicación detecta que el último instante de tiempo sea mayor a 6 min.

Último Instante de tiempo	-	-	Cada 5 min.	-
Último valor CO_2	-	Cada iteración del bucle principal	Cada vez que se actualiza el nodo en la base de datos.	Cuando el sensor se encuentra inactivo.
Último valor temperatura	-	Cada iteración del bucle principal	Cada vez que se actualiza el nodo en la base de datos.	Cuando el sensor se encuentra inactivo.
Último valor humedad	-	Cada iteración del bucle principal	Cada vez que se actualiza el nodo en la base de datos.	Cuando el sensor se encuentra inactivo.
Actuador	Cada iteración del bucle principal.	Cuando el modo es automático.	Cada vez que se actualiza el nodo en la base de datos.	Cuando el modo es manual. Cuando el sensor se encuentra inactivo.
Modo automático	Cada iteración del bucle principal.	Una única vez al registrar el nodo en la base de datos	Cada vez que se actualiza el nodo en la base de datos.	Cada vez que el usuario lo solicite.
Límite de CO_2	Cada iteración del bucle principal.	Una única vez al registrar el nodo en la base de datos	Cada vez que se actualiza el nodo en la base de datos.	Cada vez que el usuario lo solicite.
Límite de temperatura	Cada iteración del bucle principal.	Una única vez al registrar el nodo en la base de datos	Cada vez que se actualiza el nodo en la base de datos.	Cada vez que el usuario lo solicite.
Límite de humedad	Cada iteración del bucle principal.	Una única vez al registrar el nodo en la base de datos	Cada vez que se actualiza el nodo en la base de datos.	Cada vez que el usuario lo solicite.

Se puede observar que las únicas dependencias de datos que existen se encuentran en las propiedades de los límites. Estos son el actuador, el modo automático y los límites de CO_2, temperatura y humedad.

Sin embargo, gracias a que las operaciones de lectura y escritura en la base de datos son atómicas, sin importar el orden en el que se lean o escriban, tanto el nodo como la aplicación cumplen un funcionamiento correcto.

Es verdad que puede existir inconsistencia entre los dos componentes, pero al no ser un sistema crítico, no supone un problema. Además, en este caso, el nodo actualiza su propio estado en la siguiente iteración y resuelve la inconsistencia.

Cabe mencionar que la frecuencia de cinco minutos aplicada al nodo en este proyecto no es fija ni representa un límite. La frecuencia máxima soportada por el nodo es de 20 segundos, que, en caso de aplicarla, la inconsistencia se reduciría a esos 20 segundos.

Definidos los tres conceptos clave, a continuación, se explican detalladamente el diseño de los componentes del sistema.

4.1. Nodo

El diseño del nodo se basa en el patrón *maestro-esclavo*, típico de los sistemas de ejecución en paralelo, aunque presenta pequeñas diferencias siendo la más importante la ejecución secuencial.

Para diseñar el nodo, se ha hecho uso del entorno *Arduino* IDE usando el lenguaje de programación C. Todos los programas basados en este entorno tienen una clase principal compuesta por un método *setup* y un método *loop* y pueden existir varias clases de tipo *header*.

En este proyecto, la clase principal es el maestro y las clases con extensión *header* son los esclavos. El conjunto de clases definidas para el nodo es el siguiente.

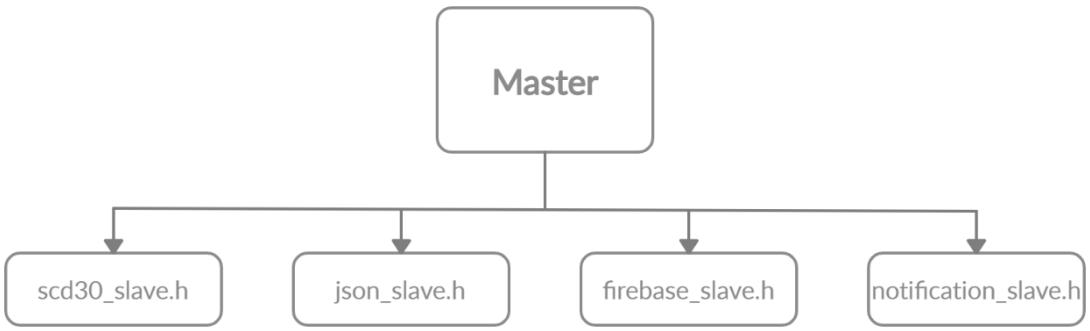


Figura 7. Arquitectura del nodo.

Clase	Descripción
Master	Se encarga de gestionar todas las funciones que debe hacer el nodo.
scd30_slave.h	Se encarga de comunicarse con el sensor y obtener los datos de CO_2, temperatura y humedad.
json_slave.h	Se encarga de convertir los datos en formato *JSON*.
*Firebase*_slave.h	Se encarga de comunicarse con la base de datos en la nube para las operaciones de lectura y escritura.
notification_slave.h	Se encarga de comprobar si los datos superan los límites establecidos. También se encarga de manejar el actuador y comunicarse con el servicio de mensajería en la nube.

4.1.1. Conexión física del nodo.

Para que la placa y el sensor puedan trabajar juntos, estos deben poder comunicarse. El nodo de este proyecto utiliza el protocolo I^2C[22] para la comunicación entre la placa y el sensor. La relación de pines es la siguiente.

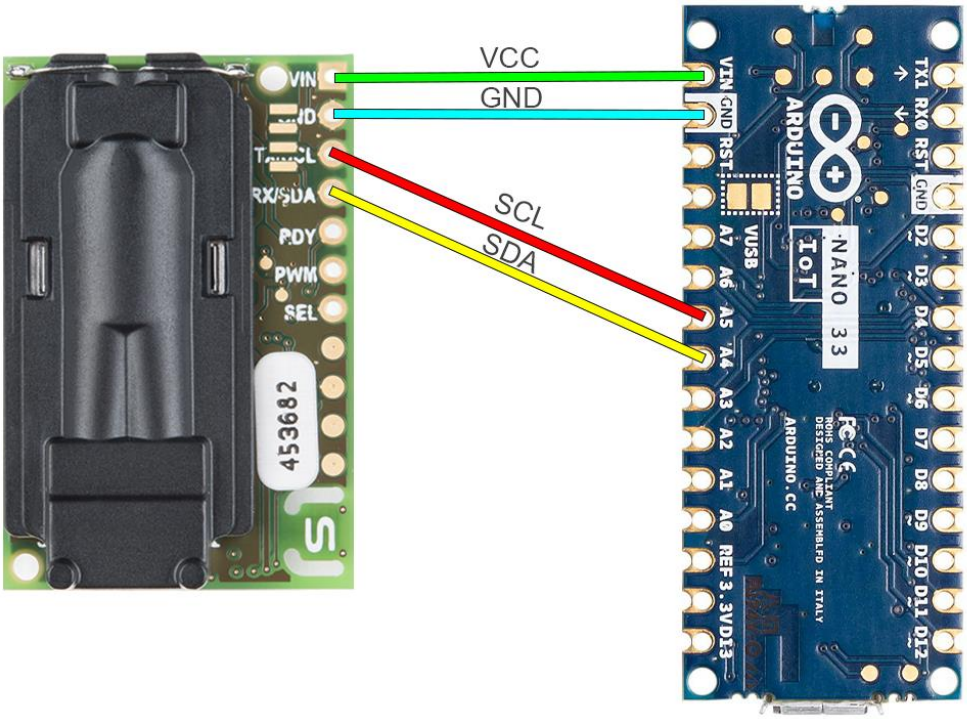


Figura 8. Conexión física del nodo.

Pin	Descripción
VCC	Voltaje I^2C, normalmente entre 1.2V y 5.5V
GND	Toma a tierra
SCL	Bus de datos (línea de datos I^2C)
SDA	Bus de reloj (línea de reloj I^2C)

[22] http://www.i2c-bus.org/de/i2c-bus

4.1.2. Librerías utilizadas

Para el correcto funcionamiento del nodo, se necesitan siete librerías que se describen a continuación.

Librería	**Descripción**
WiFi	Módulo para la comunicación vía WiFi
WiFiNINA	Módulo para la comunicación inalámbrica vía WiFiNINA
SSLClient by Noah Koontz	Módulo para establecer comunicaciones HTTPS.
SparkFun SCD30 *Arduino* Library	Módulo para la comunicación con el sensor
*Arduino*Json by Benoit Blanchon	Módulo para convertir/revertir datos en/de formato JSON.
Firebase Arduino based on WiFiNINA	Módulo para comunicar el nodo con la base de datos en la nube.
Adafruit SleepyDog	Módulo para controlar el funcionamiento de la placa.

4.1.3. Modelos

En el diseño del nodo, se utilizan dos modelos para la comunicación entre el maestro y los esclavos. La primera clase consta de tres variables que definen los valores de CO_2, temperatura y humedad, mientras que, la segunda, consta de cinco variables que definen el modo automático, el estado del actuador y los límites de CO_2, temperatura y humedad.

Data_record.h

Variable	**Tipo de datos**
val_co2	uint16_t
val_temperatura	float
val_humedad	float

Sensor_limits.h

Variable	**Tipo de datos**
lim_co2	uint16_t
lim _temperatura	float
lim _humedad	float
modo_auto	uint8_t
estado_actuador	uint8_t

4.1.4. Método *setup*

La función *setup* de la clase principal, el maestro, contiene las llamadas necesarias para inicializar los esclavos que lo necesiten y para establecer su estado en la base de datos.

En este caso, se necesita inicializar el actuador, la librería del sensor *SparkFun SCD30* y la librería *Firebase Arduino* para la comunicación con la base de datos. Para ello, tanto el esclavo *scd30_slave* como *Firebase_slave* tiene un método inicializar en el que llama a sus respectivas librerías.

La librería del sensor no necesita parámetros para inicializarse, sin embargo, la librería de *Firebase* sí. Estos parámetros son el *SSID* y contraseña *WiFi*, el host[23] del proyecto *Firebase* y la clave secreta del servidor de la base de datos.

Por último, el maestro, en este método, llama al esclavo *Firebase_slave* con el propósito de registrar el estado del sensor en la base de datos. En dicha llamada, el esclavo se conecta con la base de datos y busca la referencia de su propio nodo. En caso de ser encontrado, se actualizan los límites del nodo y en caso contrario, se introduce un registro del nodo con parámetros por defecto.

4.1.5. Método *loop*

La función *loop* del maestro se ejecuta de forma constante en intervalos de cinco minutos. Para ello, este método, debe contener todas las llamadas necesarias para actualizar el estado del nodo y enviar tanto los datos como las notificaciones cuando corresponda.

El primer paso que se realiza en esta función es la obtención de datos del sensor. El esclavo *scd30_slave* tiene un método para obtener los niveles de CO_2, temperatura y humedad. El maestro llama a dicha función pasándole una referencia del modelo *data_record* para que el esclavo lo rellene con los datos obtenidos.

Una vez hecho eso, el maestro llama al esclavo *json_slave* pasando el objeto *data_record* y una referencia a una variable de tipo *lista de cadenas* para que éste la rellene. Las cadenas de texto en formato *JSON* que genera el esclavo son cuatro. Las

[23] [NOMBRE DEL PROYECTO].firebaseio.com

primeras tres están formadas por las claves *timestamp* y *value*, cuyos valores son *.sv* para indicar que la base de datos tiene que añadir el instante de tiempo y el valor obtenido de cada tipo. La cuarta cadena está formada por un *timestamp* y los tres valores juntos. Este último texto, se usa para establecer los últimos valores del estado del nodo.

Cuando el maestro recibe las cadenas de texto, se pasan al esclavo *Firebase_slave* para que las envíe a la base de datos. Además, las cuatro llamadas efectuadas para cada una de las cadenas también contienen las rutas donde se tienen que enviar cada una.

Cabe mencionar, que los valores individuales se añaden a la base de datos a través de un método *push*, que permite que la propia base establezca una nueva clave aleatoria dentro de la ruta propuesta. En cambio, los últimos valores se sobrescriben en la ruta del nodo a través de un método *set*.

Por último, el maestro llama de nuevo al esclavo *Firebase_slave* con el objetivo de poner el estado del nodo en activo.

El segundo proceso que se realiza es actualizar el nodo. Para ello, el maestro llama otra vez a *Firebase_slave* pasándole como referencia el modelo *sensor_limits* con el objetivo de que el esclavo obtenga los parámetros del nodo y rellene el modelo.

Una vez obtenidos los parámetros, se envían los dos modelos como referencia al esclavo *notification_slave*. El esclavo, se encarga de comparar los datos obtenidos por el sensor con los parámetros actualizados. Si alguno de los valores es mayor que el límite establecido, se envía una notificación a la aplicación utilizando el servicio de mensajería y posteriormente, en relación del modo automático, se activa o desactiva el actuador.

La comunicación con el servicio de mensajería se realiza a través de una conexión segura utilizando el protocolo *HTTPS*. Los parámetros que se usan para esta comunicación son los siguientes.

Parámetro	Valor
Host	fcm.googleapis.com
Port	443
POST	/fcm/send HTTP/1.1
Authorization	key=AUTH_KEY
Content-Type	application/json
Content-Length	TAMAÑO_DATOS

El valor *AUTH_KEY* hace referencia a la clave secreta del servidor de mensajería de *Firebase*. Dicha clave, se encuentra en los ajustes del proyecto, en la sección de mensajería en la nube, como se indica en la solución propuesta del análisis del problema.

Los datos que se envían en el cuerpo de la solicitud están en formato *JSON* y son los siguientes.

Clave	Valor
to	TOKEN_ID
notification/title	Título de la notificación.
notification/body	Cuerpo de la notificación.
data/ref	Referencia del nodo.
data/tipo	Tipo de dato (CO_2, temperatura, humedad)
data/value	Valor obtenido.
data/auto	Modo en el que se encuentra el nodo (automático/manual).
data/click_action	*FLUTTER*_NOTIFICATION_CLICK

La primera clave tiene como valor un *TOKEN_ID* del dispositivo móvil al que se quiere enviar la notificación. Se ve como obtener dicho valor en el apartado *4.3.3 — Notificaciones*. Las siguientes dos claves son el título y el cuerpo que aparecen en la notificación cuando la recibe la aplicación. Dentro de la clave *data*, se definen cuatro más incluyendo la referencia del nodo, el tipo de datos, el valor y el modo de funcionamiento del nodo. Por último, es necesario definir la clave *click_action* dentro de *data* con el valor que se indica, para que la aplicación pueda abrirse al presionar la notificación, cuando esta esté cerrada o en segundo plano.

El actuador se activa o desactiva en función de tres variables. La primera de ellas indica si los límites del nodo han sido rebasados, la segunda indica el modo de funcionamiento y la tercera indica si el actuador debe estar activado o desactivado. A continuación, se muestra una tabla de verdad con las situaciones posibles.

Límites superados	Modo	Actuar	Actuador
-	Manual	Si	Activar
-	Manual	No	Desactivar
Si	Automático	-	Activar
No	Automático	-	Desactivar

El tercer y último paso que se realiza es poner la placa en modo inactivo durante cinco minutos para ofrecer un bajo consumo. Con ayuda de la librería *Adafruit SleepyDog*, se llama a la función *sleep* pasándole 1000 milisegundos como tiempo, ya que es lo máximo que permite la placa. Este proceso se repite mientras los milisegundos totales sean menores que 300.000. Al salir de ese bucle, se vuelve a ejecutar la función *loop* de nuevo.

4.2. Plataforma *IoT*

Como se ha comentado anteriormente, la base de datos de *Firebase* es no relacional y presenta una estructura JSON, siendo esta, un formato constituido por pares clave-valor.

Los datos se dividen en dos rutas principales. En la siguiente figura se muestra un esquema de estas dos rutas.

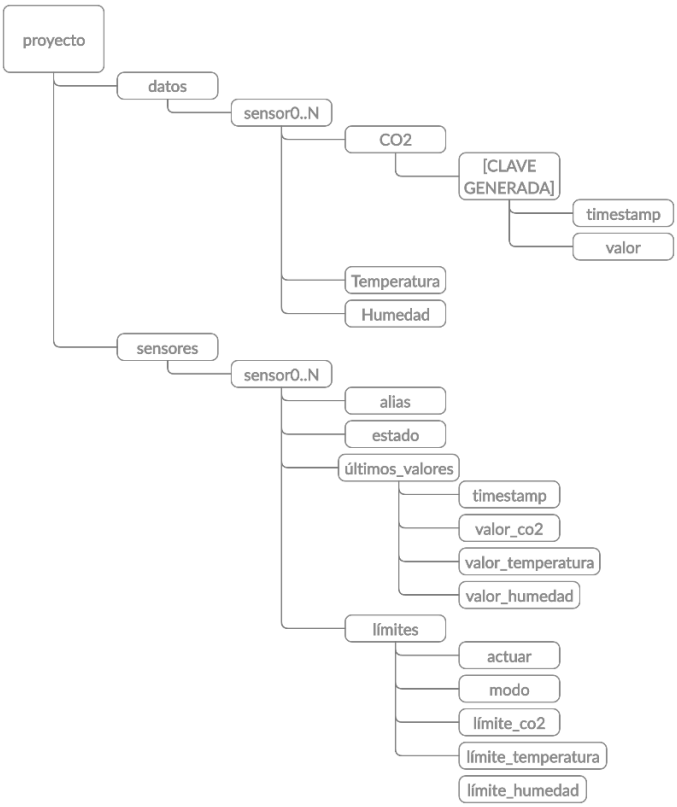


Figura 9. Rutas de la base de datos.

Las dos rutas tienen como valores las referencias de los nodos.

En la ruta de datos, cada uno de los nodos tiene tres apartados referenciando los tipos de datos que capturan, es decir, CO_2, temperatura y humedad. Dentro de cada tipo se registran los datos capturados por el nodo referenciado.

Cada uno de estos registros, cuya clave se genera aleatoriamente en la base de datos, contiene a su vez un instante de tiempo, que también se genera en la base de datos indicando el instante del registro en formato *Unix Time Stamp*[24] y el valor correspondiente.

En la ruta de los sensores, cada nodo contiene los elementos que forman su estado siendo estos el *alias*, el *estado*, los *últimos valores* y los *límites*. A su misma vez, el elemento *últimos valores* contiene un instante de tiempo y los valores de CO_2, temperatura y humedad y el elemento *límites* está formado por las claves actuar, modo, límite de CO_2, límite de temperatura y límite de humedad.

[24] https://www.unixtimestamp.com/

4.3. Aplicación móvil

El diseño de la aplicación móvil se realiza en *Flutter*, una herramienta basada en el lenguaje de programación *Dart*[25]. La arquitectura de las aplicaciones desarrolladas en *Flutter* están basadas en *Widgets*, siendo estos la unidad principal de visualización de elementos. Cada uno de los componentes que se pueden ver en las diferentes vistas de una aplicación son *Widgets* que, a su misma vez, están compuestos por otros *Widgets*, formando así un esquema en forma de árbol donde cada pieza es un *Widget*.

La estructura de la aplicación móvil sigue dos esquemas principales. El primero de ellos y el más general es la división en capas de presentación, lógica y datos, mientras que, el segundo, se basa en la arquitectura *MVVM*, un patrón creado por Microsoft. En este segundo esquema, la división se basa en tres componentes principales siendo estos, las vistas, los modelos y los modelos de las vistas. Juntando los dos esquemas, se crea uno nuevo en el que:

- La capa de presentación está formada por las vistas.
- La capa de lógica está formada por los modelos, los modelos de las vistas y los repositorios.
- La capa de datos está formada por un servicio para contactar con la base de datos.

En la siguiente figura se puede observar el resultado.

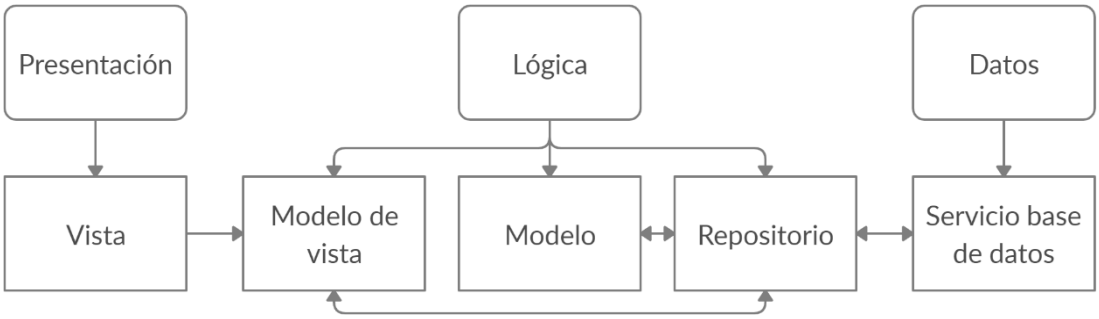


Figura 10. Arquitectura de la aplicación móvil.

[25] https://dart.dev/

4.3.1. Pantalla de inicio

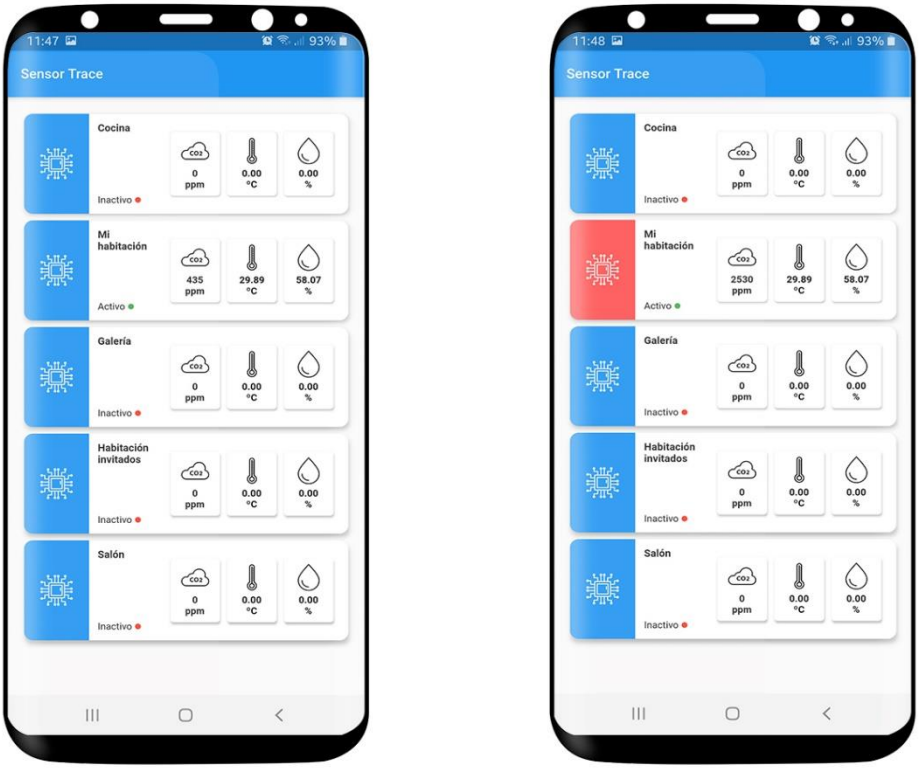


Figura 11. Pantalla de inicio.

La pantalla de inicio tiene una lista como elemento principal donde cada entrada representa un nodo registrado en la base de datos.

Entre las propiedades que conforman el estado del nodo, en cada entrada, se muestran el alias, el estado del nodo y sus últimos valores. También se indica mediante un cambio de color, si alguno de los últimos valores supera su correspondiente límite.

La representación de la vista actual es producto de una lista de elementos de tipo *estado de nodo*, que mantiene su modelo de vista correspondiente. Para que el modelo de vista rellene dicha lista, llama a su repositorio. El repositorio, a su vez, llama al servicio de datos para que devuelva una conexión abierta apuntando a la ruta de nodos. De esta manera, una primera ráfaga de datos le llega al repositorio con todo el

contenido de la ruta en formato *JSON* que el propio repositorio transforma en una lista de objetos de tipo “estado de nodo” y se lo pasa al modelo de vista. Cuando el modelo de vista actualiza el contenido de su propia lista, inmediatamente la vista cambia su estado mostrando el nuevo contenido.

A partir de esta primera ráfaga, cada vez que se actualiza la base de datos en la ruta de los nodos, el repositorio obtiene una ráfaga con el contenido en tiempo real y se repite el proceso manteniendo así la vista con los datos actualizados.

Otra de las funcionalidades que tiene el repositorio de esta vista consiste en un temporizador, que cada cinco minutos le indica al servicio de base de datos que le pase la lista actualizada de nodos. Una vez obtenida, recorre dicha lista y comprueba el último instante de tiempo en que el nodo añadió datos. En caso de que el instante de tiempo sea mayor a seis minutos, pide al servicio que ponga el nodo en estado inactivo.

4.3.2. Pantalla de nodo

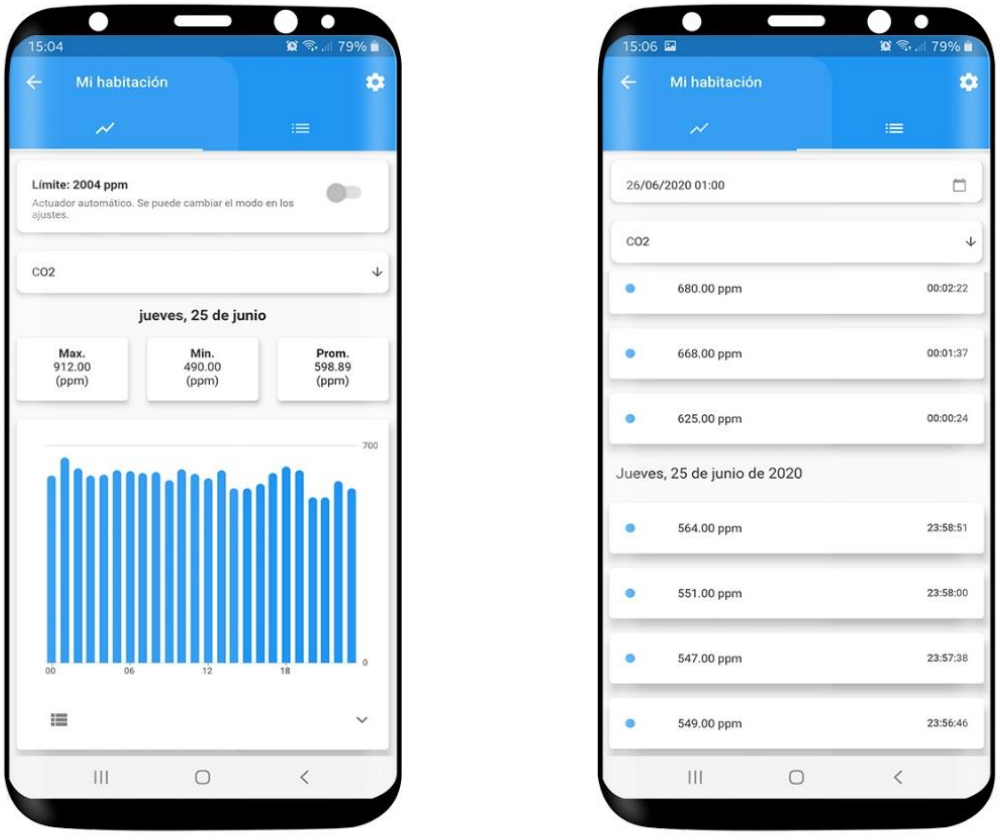


Figura 12. Pantalla de nodo.

Al pulsar sobre alguna de las entradas de la lista de la pantalla principal, una segunda pantalla se abre enfocándose en tratar con el nodo seleccionado.

La vista está formada por un botón situado en la barra superior permitiendo ir a la vista de los ajustes del nodo y dos vistas internas e intercambiables a través de un menú de pestañas. Estas vistas son nombradas *vista gráfica* y *vista lista.*

4.3.2.1. Vista gráfica

La pestaña *vista gráfica* está conformada por tres elementos principales. Estos son una tarjeta situada en la parte superior indicando el límite para el tipo de datos seleccionado, el modo de funcionamiento y el estado del actuador. Otra tarjeta por debajo que permite seleccionar el tipo de datos a visualizar. Por último, un panel deslizable con un conjunto de elementos en el que se muestran los datos de un día en concreto.

A continuación, se van a explicar parte por parte los elementos.

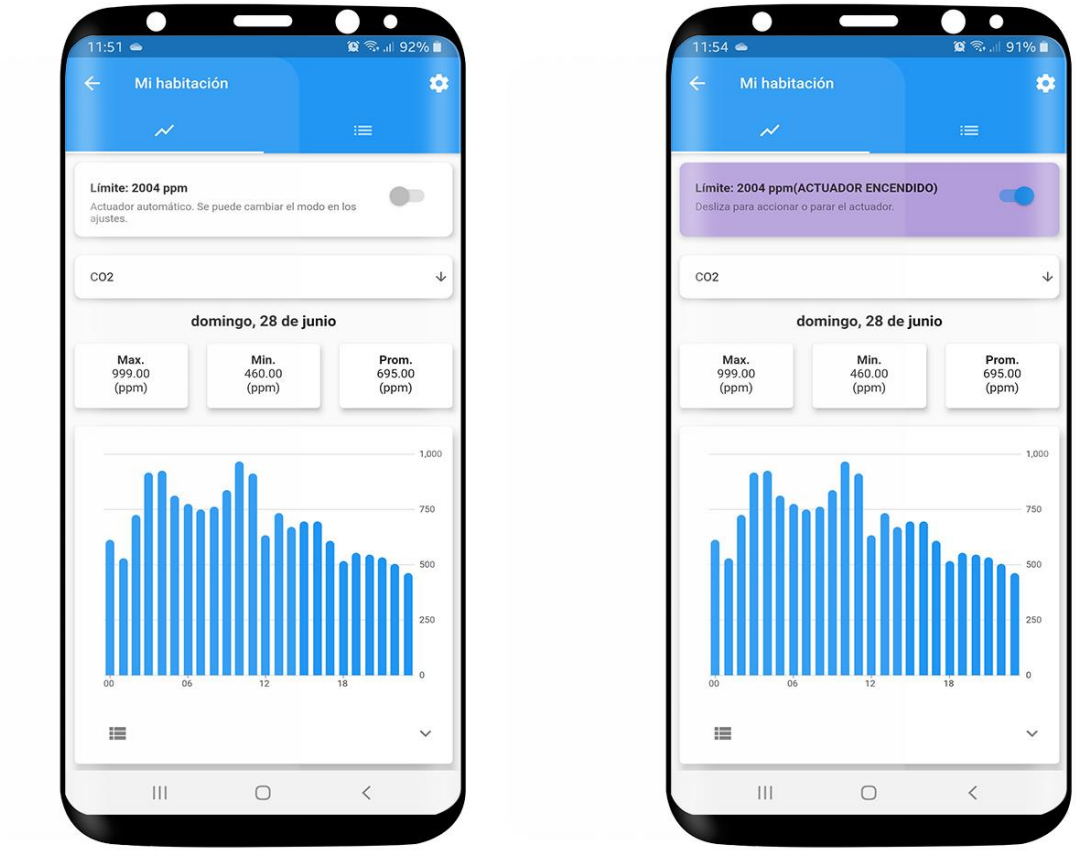


Figura 13. Tarjeta del actuador.

La tarjeta de la parte superior está formada por un título, un subtítulo y un interruptor. En el título se muestra el límite establecido para el nodo en cuanto al tipo de datos actual, en este caso CO_2. Por otro lado, el subtítulo indica el modo de funcionamiento y una breve información acerca de las acciones que el usuario puede realizar. Finalmente, el interruptor concuerda con el estado del actuador y únicamente se puede controlar cuando el modo de funcionamiento sea manual.

La tarjeta cambia de color para mostrar cuando el actuador este encendido, al igual que el título cambia indicándolo. Se puede observar el cambio en la Figura 13.

Para lograr este comportamiento, un modelo de datos contiene una variable con el estado del nodo que se actualiza al paso con la base de datos. Esto es así debido a una conexión abierta sobre la ruta del nodo, que el repositorio de la actual vista pide al servicio de la base de datos y la mantiene.

Cuando el modo de funcionamiento es automático, el interruptor permanece desactivado para que el usuario no lo pueda manejar ya que ese es el propósito del modo automático, que el propio nodo sea quien maneja el actuador. Este modo se puede cambiar en los ajustes del nodo, vista que se explica en el apartado *4.3.2.3 — Vista de ajustes*.

Cuando el modo de funcionamiento es manual, el interruptor se activa permitiendo que el usuario sea quien maneja el actuador. Debido a que el nodo realiza un ciclo cada cinco minutos, el tiempo máximo entre la acción del usuario y la activación o desactivación del actuador es de esos cinco minutos. Si durante ese periodo el usuario activa y desactiva el actuador en repetidas ocasiones, el nodo se queda con el valor establecido en el momento de la lectura del parámetro, invalidando todas las acciones anteriores a dicha lectura.

Tanto en el modo automático como el manual, el color de la tarjeta sigue indicando si el actuador está encendido. No ocurre lo mismo con el interruptor, ya que, al estar desactivado en el modo automático, no se puede apreciar con total claridad si el actuador está activo o inactivo.

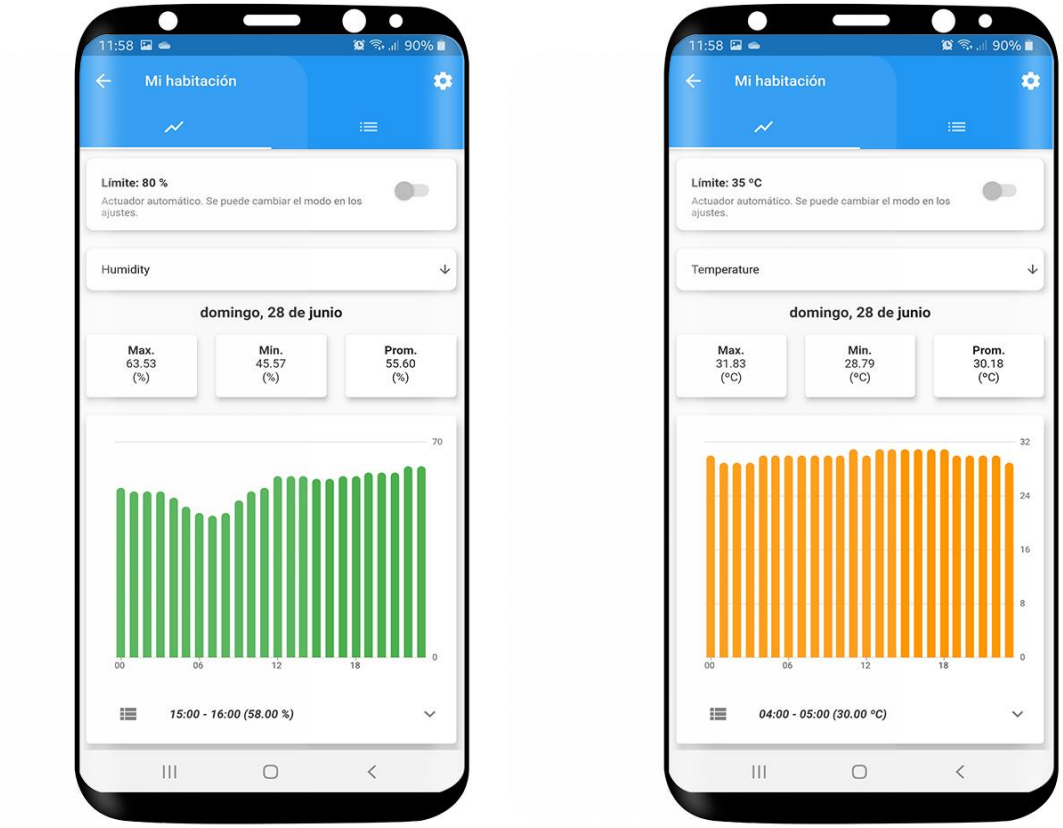


Figura 14. Tarjeta de tipos.

La tarjeta situada por debajo de la anterior indica el tipo de datos que se visualiza. Al pulsar sobre esta tarjeta, se abre un menú desplegable en el que se puede seleccionar el tipo de datos que se quiere visualizar.

El parámetro seleccionado influye directamente en el panel inferior, ya que este debe recargar su vista con los datos solicitados. Se puede observar dicha funcionalidad en la Figura 14.

Para lograr este comportamiento, el modelo de vista tiene una variable que mantiene el estado de la selección. En cuanto esta se cambia, se activa un proceso para obtener los datos correspondientes al tipo seleccionado que pasa por el repositorio hasta llegar al servicio de la base de datos.

Este último, obtiene los datos del día y el tipo en cuestión y se los devuelve al repositorio en formato *JSON*. El repositorio, transforma los datos en una lista con objetos de tipo “dato” y se los devuelve al modelo de vista. Al igual que antes, cuando dicha lista se actualiza, la vista recarga el panel con los nuevos datos.

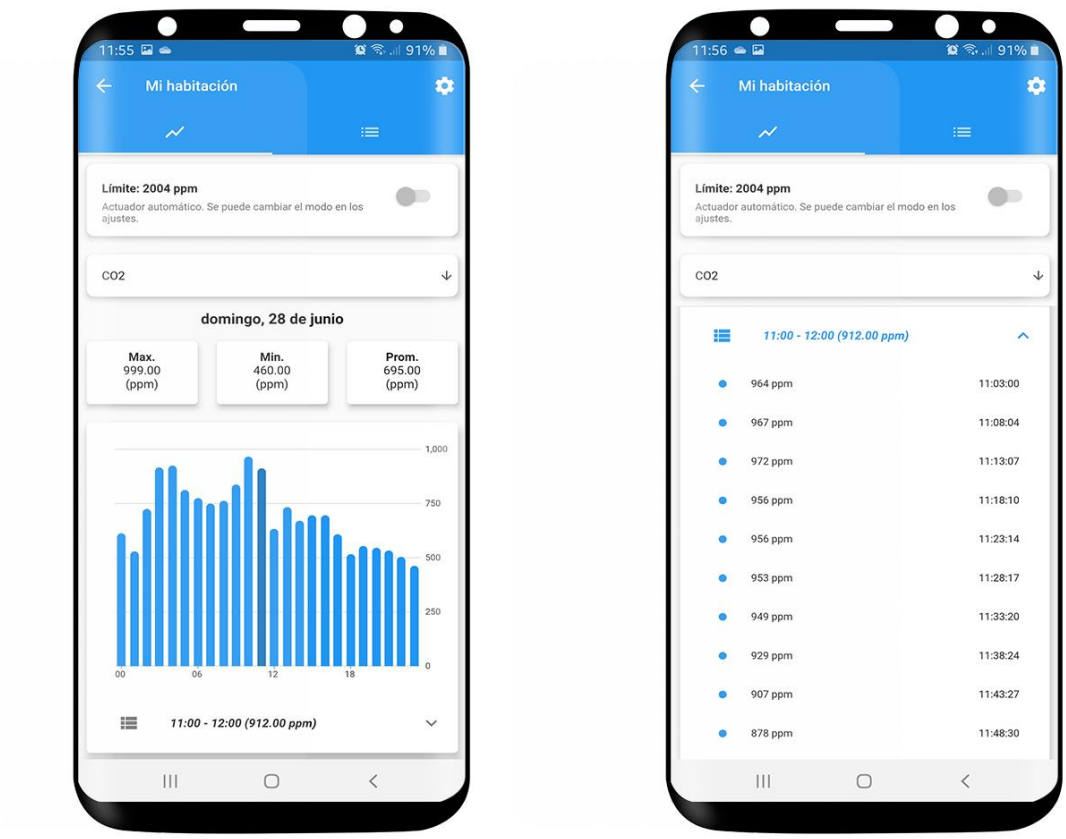


Figura 15. Panel de datos

El panel está compuesto por seis elementos que, ordenados de arriba abajo, son los siguientes.

- Un título que representa el día sobre el que se muestran los datos.
- El valor máximo, mínimo y promedio de ese día.
- Un gráfico de barras que muestra el valor medio de cada hora.
- Una lista desplegable que sirve para que el usuario, al seleccionar una de las barras del gráfico, pueda ver el valor de esa hora y una lista con los valores registrados dentro de la hora.

Este panel es deslizable horizontalmente. Cuando el usuario necesita ver los datos de otro día, puede realizar un gesto de deslizamiento sobre el panel, cambiando así de día. Cuando se desliza hacia la izquierda, se muestra el día siguiente al actual en la vista en caso de que exista, mientras que cuando se desliza hacia la derecha, se muestra el día anterior.

Una vez el usuario selecciona una hora en el gráfico de barras, el título y el contenido de la lista desplegable, se actualizan con la hora y el valor medio de esa hora y los valores registrados en esa hora respectivamente. Además, el usuario puede pulsar sobre la lista para expandirla y ver esos valores o contraerla y volver al estado normal. En la Figura 15 se pueden observar los dos estados.

El funcionamiento interno del panel está basado en una variable que controla el día seleccionado, la variable tipo de datos explicada anteriormente, tres variables que controlan el máximo, mínimo y promedio y dos listas. La primera contiene objetos de tipo dato para representar el gráfico de barras. La segunda contiene 24 listas con los valores registrados de cada hora para representar la lista desplegable en cada hora. Para que el repositorio pueda conseguir los datos, el modelo de vista le pasa la referencia del nodo, el tipo de datos y el día en cuestión.

El repositorio, llama al servicio de base de datos haciendo uso de estos parámetros para que le devuelva los datos correspondientes al tipo y al día. Este servicio, a través de la *API de Firebase*, busca los datos en la ruta de datos utilizando la referencia del nodo y el tipo. Además, la *API* proporciona tres métodos para ordenar los datos según el instante de tiempo, para especificar el límite inferior y para especificar el límite superior. Gracias a estos métodos, el servicio utiliza el día que ha recibido como parámetro para establecer dichos límites indicando que los datos se deben ordenar por instante de tiempo.

Cuando el repositorio recibe los datos, primero los transforma en objetos de tipo dato. Posteriormente, los divide en 24 listas según el instante de tiempo del dato. Por último, calcula el máximo, mínimo y el promedio y monta la lista final con las medias de cada hora para pasarlo todo al modelo de vista.

Finalmente, el modelo de vista actualiza todas las listas y estadísticas y notifica a la vista para que recargue sus elementos. Cada vez que el usuario interactúa con el panel deslizable o el tipo de datos, se repite este proceso.

4.3.2.2. Vista lista

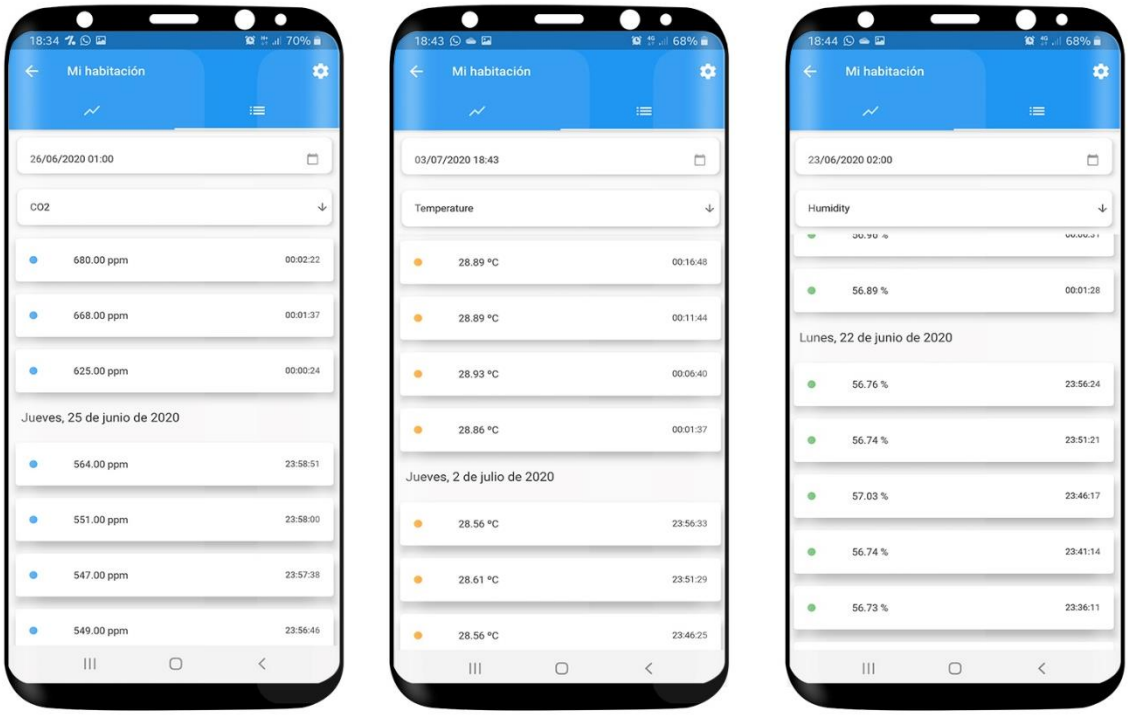


Figura 16. Listas de CO_2, temperatura y humedad.

La pestaña *vista lista* está pensada para una visualización más rápida y precisa de los datos. Al igual que la otra pestaña, esta está formada por tres elementos principales, dos tarjetas y una lista. A continuación, se detallan estos elementos.

La primera tarjeta, posicionada justo debajo del menú de pestañas, indica la fecha y hora a partir de las que se obtiene los datos. Al pulsar sobre dicha tarjeta, aparece un diálogo donde el usuario puede indicarlas. Una vez introducidas y aceptadas, se actualiza la lista de datos. En caso de cancelación, no se produce ninguna modificación.

La segunda tarjeta indica el tipo de datos a visualizar y se sitúa entre la primera tarjeta y la lista. Se muestra un menú desplegable al presionar sobre la tarjeta con las tres opciones posibles, siendo estas, CO_2, temperatura y humedad. Cuando una de ellas es seleccionada, la lista de datos se actualiza con los datos necesarios.

La lista de datos, en el primer contacto con la pestaña, carga los últimos 20 datos registrados en la base de datos. Cuando el usuario desliza dicha lista y llega a su último

elemento, carga 20 datos más. El mismo proceso se repite hasta que no queden más datos registrados. Otra característica de esta lista es la separación de los datos por días con el objetivo de ubicar al usuario.

Para el funcionamiento interno de esta pestaña se necesita definir un nuevo modelo que incluye cuatro propiedades.

Propiedad	**Descripción**
Instante de tiempo	Define el instante de tiempo en el que el dato se ha registrado en la base de datos.
Tipo	Define el tipo de dato capturado (CO_2, temperatura, humedad).
Valor	Define el valor capturado.
Grupo	Indica si el modelo es un dato por mostrar o un título.

Este nuevo modelo llamado *dato_lista* está basado en el modelo *dato* con la inclusión de una nueva propiedad llamada *grupo*. Esta propiedad tiene como objetivo diferenciar entre las entradas que contienen los valores de los datos y los títulos mostrados al cambiar de día.

Cuando se activa la pestaña por primera vez, la fecha se ajusta al instante de activación y el tipo de datos a mostrar por defecto se fija en CO_2. Estos parámetros están definidos en dos variables en el modelo de datos. Una tercera variable de tipo lista de objetos *dato* se rellena con los datos que el repositorio le pasa. Para ello, se le llama con los parámetros mencionados y con la referencia del sensor.

El repositorio, por otro lado, mantiene una variable que define el instante de tiempo del último valor que obtiene a través de las llamadas al servicio de base de datos. Esto es así para permitir la funcionalidad de cargar 20 elementos cada vez.

Cuando el repositorio recibe una nueva fecha, ajusta su variable a la misma y llama al servicio de datos indicando la referencia del sensor, y el instante de tiempo establecido. El servicio, al igual que antes, obtiene los registros de la ruta de datos especificada por el tipo. Además, se ordena por instantes de tiempo ascendentemente con un límite inferior especificado por la fecha recibida y un límite de registros de tamaño 20. Finalmente, le devuelve una cadena en formato *JSON* que el propio repositorio convierte en una lista de datos con objetos de tipo *dato_lista*.

Finalmente, el repositorio establece su variable con el instante de tiempo del último objeto de la lista resultante y le devuelve dicha lista al modelo de vista. El modelo

de vista actualiza su propia lista con los nuevos valores obtenidos y notifica a la vista para que recargue sus elementos.

Cuando el usuario desliza y llega al final, se cargan otros 20 elementos a partir del instante definido por la variable del repositorio. En cambio, si el usuario selecciona una nueva fecha u otro tipo de datos, la variable se ajusta al instante indicado permitiendo así cargar la lista a partir de la nueva fecha.

4.3.2.3. Vista de ajustes

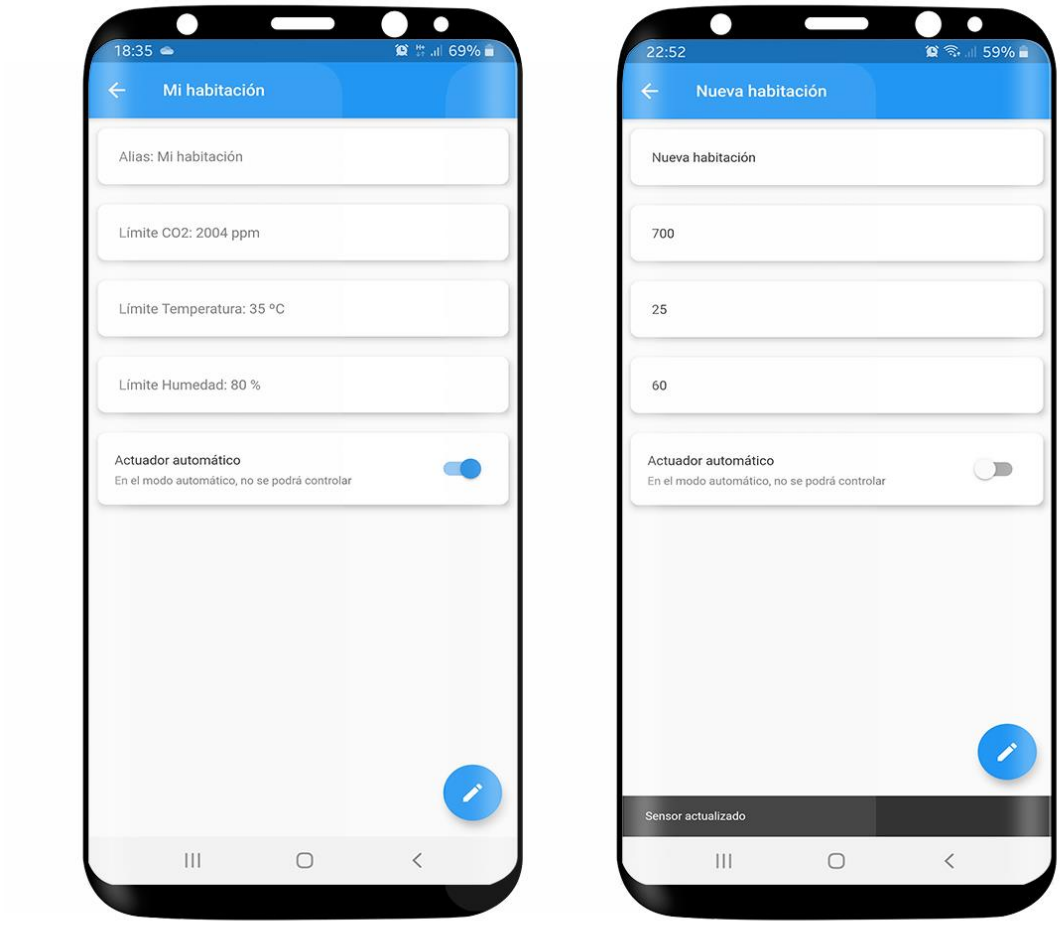


Figura 17. Vista de ajustes.

La vista de los ajustes del nodo está formada por seis elementos.

- Tarjeta con una caja de texto para establecer el alias del nodo.
- Tarjeta con una caja de texto para establecer el límite de CO_2.
- Tarjeta con una caja de texto para establecer el límite de temperatura.
- Tarjeta con una caja de texto para establecer el límite de humedad.
- Tarjeta con un interruptor para establecer el modo de funcionamiento.
- Botón para aplicar los cambios.

Cada una de las tarjetas representa una propiedad modificable del estado del nodo. Además, dentro de cada caja, existe un texto prestablecido, o el estado del interruptor en su caso, indicando los valores actuales de dichas propiedades.

Las modificaciones se pueden realizar de forma individual, es decir, se puede cambiar cualquier propiedad sin necesidad de que todas las cajas de texto estén rellenadas.

Una vez se establecen los campos que se quieren modificar, se pulsa el botón para que se apliquen los cambios y así actualizar la base de datos.

Este funcionamiento se logra a través de un modelo de vista que contiene una variable para cada propiedad mostrada. Cuando el botón se pulsa, el modelo envía las propiedades y la referencia del nodo al repositorio para que este llame al servicio de la base de datos y actualice el nodo. Antes de esta última llamada, el repositorio le pide al servicio obtener el estado del nodo por si durante este proceso, el nodo ha cambiado algún parámetro de su estado como por ejemplo los últimos valores registrados.

Al finalizar la actualización, el estado del nodo mantenido en el modelo de la vista también se actualiza con los nuevos valores. Además, se muestra una notificación en pantalla indicando que los cambios se han aplicado.

4.3.3. Notificaciones

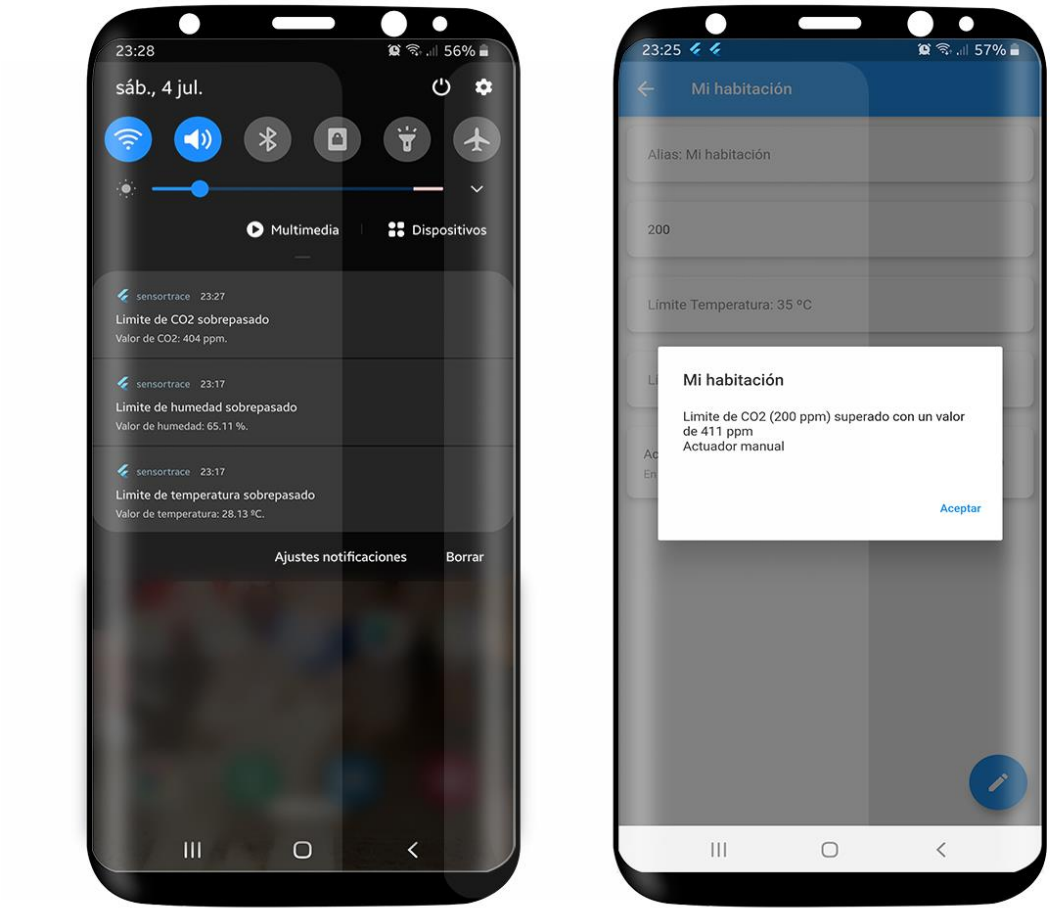


Figura 18. Vista de notificaciones.

Para que el dispositivo pueda recibir notificaciones de la aplicación, además de necesitar el *SDK de Firebase* como se comentó en el apartado *3.6 — Solución propuesta*, debe cumplir con dos requisitos.

El primer requisito es tener presente la librería *Firebase_messaging*[26] en el proyecto.

El segundo requisito es instanciar en el *main* del proyecto una clase parecida a la siguiente.

[26] https://pub.dev/packages/firebase_messaging

```dart
import 'package:firebase_messaging/firebase_messaging.dart';
class PushNotificationProvider {
  FirebaseMessaging _firebaseMessaging = FirebaseMessaging();
  initNotifications() {
    _firebaseMessaging.requestNotificationPermissions();
    _firebaseMessaging.getToken().then((value) {
      print("======= FCM Token =======");
      print(value);
    });
    _firebaseMessaging.configure(
      //APP EN PRIMER PLANO
      onMessage: (info) async {
      },
      //APP CERRADA
      onLaunch: (info) {
      },
      //APP EN SEGUNDO PLANO
      onResume: (info) {
      },
    );
  }
}
```

Figura 19. Clase de Firebase Messaging.

Esta clase permite obtener el *TOKEN_ID*, necesario para que el nodo pueda enviar las notificaciones al dispositivo y permite manejar tres situaciones distintas.

La primera situación se presenta cuando la aplicación esta activa en pantalla y llega una notificación. El manejo viene dado por el método *onMessage*.

La segunda situación se presenta cuando la aplicación esta inactiva, es decir, en segundo plano y llega una notificación. Su manejo viene dado por el método *onResume*.

La tercera y última situación se presenta cuando la aplicación está cerrada y llega una notificación. Se maneja a través del método *onLaunch*.

En este proyecto, se ha optado por manejar las tres situaciones de la misma manera, siendo esta, mostrar un diálogo que tiene como título el alias del nodo y como cuerpo un texto en el que se indica el tipo de dato sobrepasado, así como su límite y el valor que ha provocado el rebaso. También se indica el modo de funcionamiento establecido.

En caso de que la aplicación esté cerrada o en segundo plano, el dispositivo recibe una notificación indicando el tipo de datos y el valor que ha superado el límite. Al

pulsar la notificación, se abre la aplicación mostrando el diálogo establecido. En la Figura 19 se pueden apreciar los dos casos.

5. Desarrollo de la solución propuesta

El diseño del nodo, de la plataforma *IoT* y de la aplicación móvil están planteados para formar un conjunto obteniendo así un sistema completo y funcional. A continuación, se detallan las principales funcionalidades que ofrece este sistema.

5.1. Obtención y visualización de datos

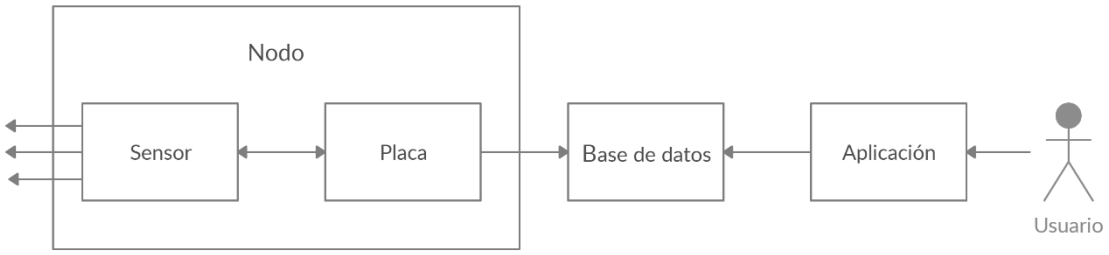


Figura 20. Obtención y visualización de datos.

La principal funcionalidad del sistema es la de obtener los datos a través del nodo y mostrarlos en la aplicación para que el usuario las pueda visualizar.

Este proceso comienza por el sensor, obteniendo los datos de CO_2, temperatura y humedad. Una vez capturados, se los envía a la placa para que este, a su vez, los envíe a la base de datos.

Por otro lado, el usuario, al acceder a la aplicación, puede visualizar dichos datos tanto en la pantalla principal en caso de que sean los últimos, como en la pantalla del nodo correspondiente a través de la pestaña con la gráfica o la pestaña con la lista.

5.2. Modo automático

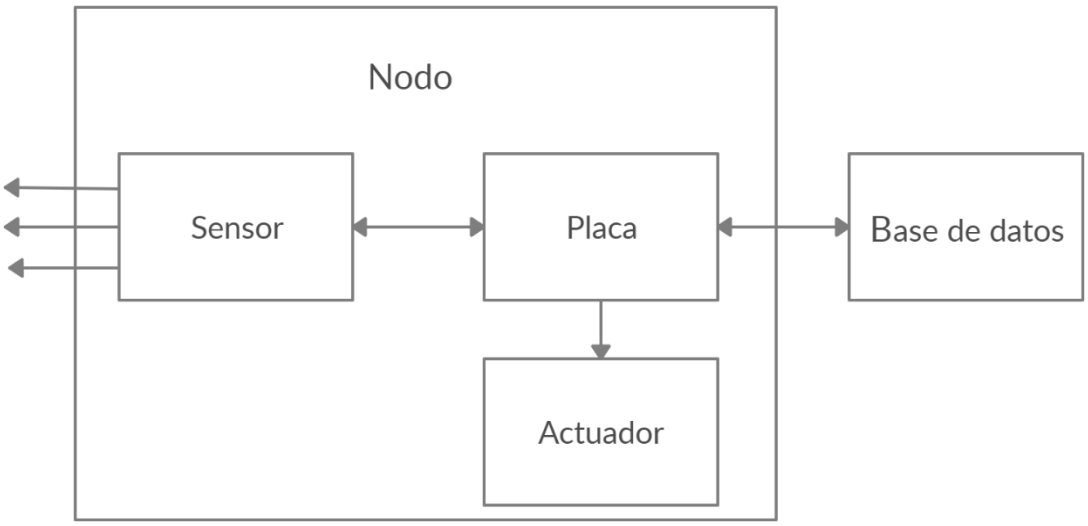


Figura 21. Modo automático.

Otra de las funcionalidades que tiene el sistema, es la capacidad de manejar el actuador de manera automática.

El desarrollo que sigue este modo comienza cuando el nodo obtiene el CO_2, la temperatura y la humedad. Posteriormente, el nodo actualiza los límites de cada dato y los compara con los obtenidos. En caso de que alguno de estos datos supere su límite, el propio nodo activa el actuador, mientras que, si ninguno de los límites se rebasa, lo desactiva.

5.3. Modo manual

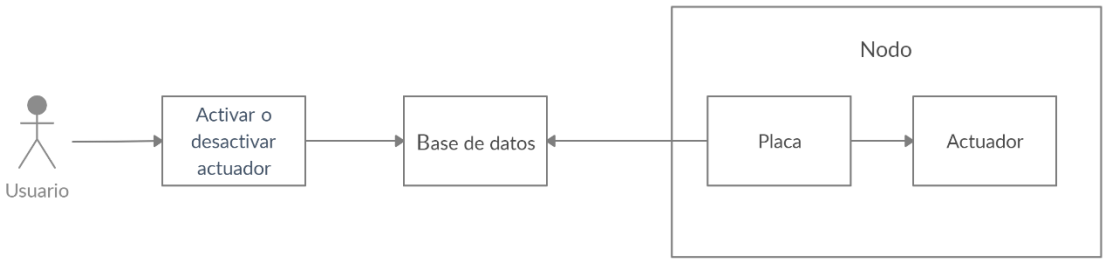


Figura 22. Modo manual.

El funcionamiento manual requiere la acción del usuario, al contrario que el modo automático.

Esta funcionalidad inicia cuando el usuario, desde la aplicación, activa o desactiva el interruptor que mantiene el estado del actuador de un nodo determinado. Conforme se produce dicha acción, el parámetro *actuar* de la base de datos se actualiza con el nuevo valor.

Por otro lado, cuando el nodo procede con la lectura de su estado sobre la base de datos, obtiene el valor del parámetro mencionado y, sin tener en cuenta si los límites han sido sobrepasados o no, procede con la activación o desactivación de su actuador.

5.4. Notificaciones

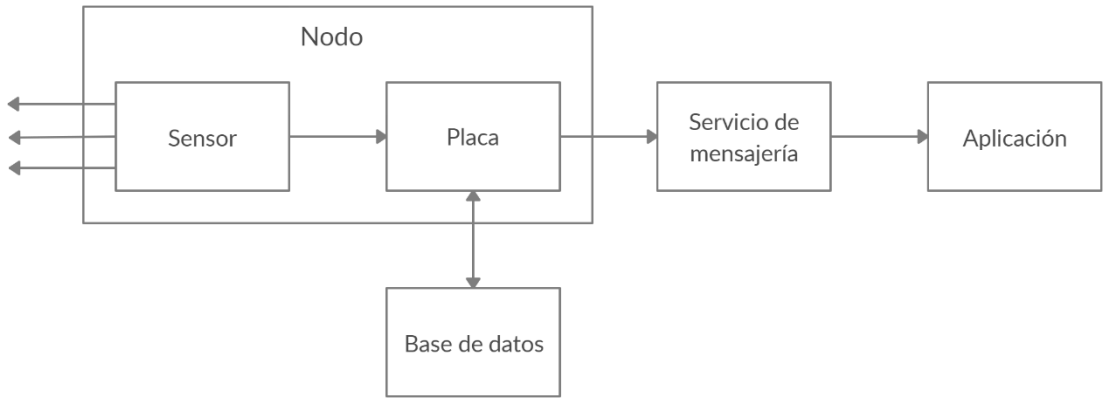


Figura 23. Notificaciones.

La última funcionalidad que posee este sistema consiste en el envío de notificaciones.

Al igual que antes, el desarrollo comienza con la obtención de los datos del aire y sigue con la actualización del estado del nodo a través de la base de datos. A la hora de comparar los datos obtenidos con los límites correspondientes, si alguno de ellos se ve superado, el nodo envía una petición al servicio de mensajería para que este envíe una notificación al dispositivo móvil. Si la aplicación se encuentra activa en el momento de la recepción, se muestra un diálogo indicando el nodo donde se ha producido el rebaso, así como el valor, el tipo de dato y su límite. De otra manera, si la aplicación está inactiva o cerrada, se muestra una notificación en el panel de notificaciones que, al ser pulsada, entra en dicha aplicación y se muestra el diálogo de la misma manera.

6. Implantación

Para la implantación del sistema existen dos requisitos principales.

El primero de ellos es un ordenador con sistema operativo *Windows 10* en cualquier versión que tenga instalado el programa *Arduino IDE* versión *1.8.12* o superior. La placa se conecta al ordenador vía *USB* y se modifican los parámetros para la conexión *WiFi* en el programa. Una vez hecho, basta con subir dicho programa a la placa.

El segundo requisito es un dispositivo *Android* real o emulado con una versión *4.1.x* o superior. Además, este dispositivo debe poder conectarse a internet para el correcto funcionamiento de la aplicación.

7. Pruebas

A lo largo del desarrollo del proyecto, se han realizado diferentes pruebas para comprobar el correcto funcionamiento de los componentes.

7.1 Nodo

Las pruebas realizadas en el nodo se definen por las diferentes funcionalidades que éste debe ofrecer. Para ello, se ha comprobado que cada uno de los esclavos de la arquitectura del nodo funciona correctamente.

Cabe mencionar, que no se han podido manejar las pruebas de consumo debido a la falta de instrumental.

En primer lugar, se ha comprobado que la obtención de datos del sensor a través del protocolo I^2C funciona correctamente. Para ello, se han incluido mensajes en la consola con los valores obtenidos.

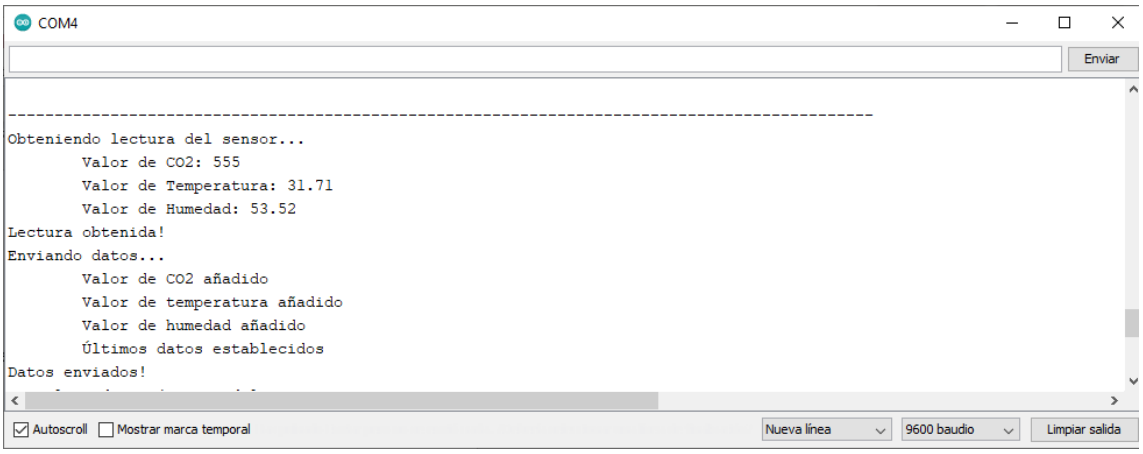


Figura 24. Prueba de obtención de datos.

En la Figura 24 se pueden observar los valores de CO_2, temperatura y humedad capturados, siendo estos 555 partes por millón (ppm), 31.71 grados Celsius (ºC) y 53.52 por ciento (%), respectivamente.

En segundo lugar, se ha comprobado que la lectura y escritura en la base de datos funciona correctamente. Dichas comprobaciones, incluyen también las pruebas de transformación de texto en formato JSON y viceversa.

Para realizar esta prueba, se han observado los valores que el nodo lee de la base de datos, mostrando mensajes a través de la consola y los valores que introduce en dicha base.

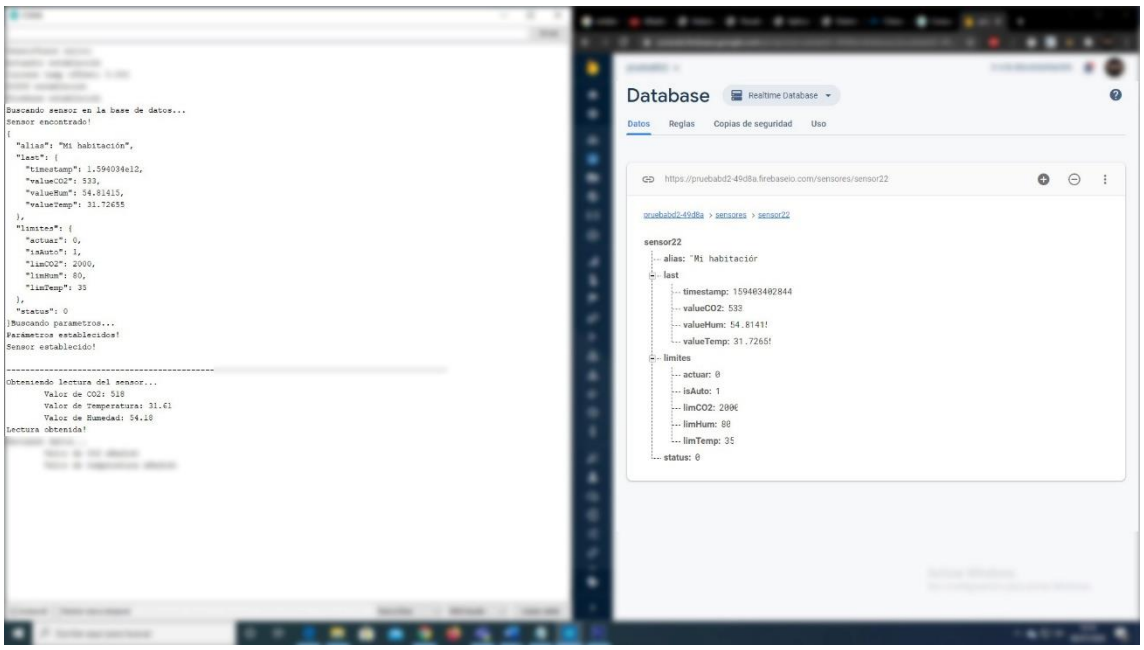


Figura 25. Prueba de lectura del nodo en la base de datos.

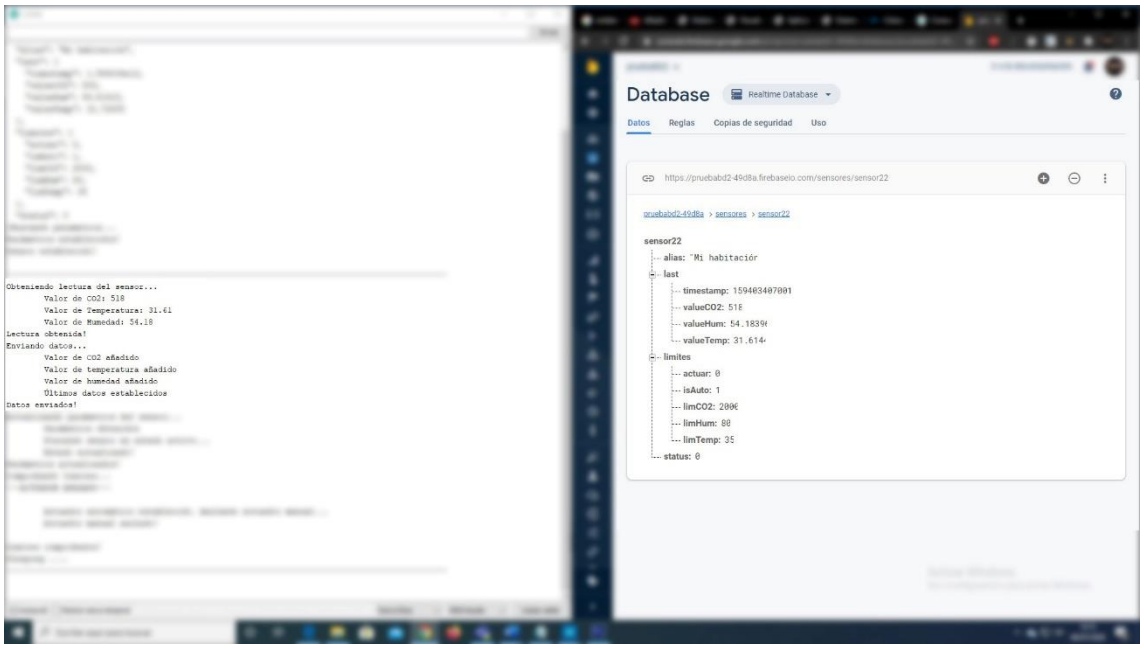


Figura 26. Prueba de escritura del nodo en la base de datos.

La Figura 25 y la Figura 26 representan la lectura y escritura respectivamente. En la primera de ellas se puede observar que la información obtenida, en la parte izquierda, corresponde con la información de la base de datos, parte derecha.

Seguidamente, en la segunda figura se puede observar como el nodo obtiene los datos, en la parte izquierda, y los envía a la base de datos, la parte derecha.

En tercer lugar, se ha comprobado el funcionamiento del servicio de mensajería. Para ello, se han establecido los límites de CO_2, temperatura y humedad a cero, con el objetivo de que el nodo detecte que dichos límites han sido sobrepasados y proceda a enviar las notificaciones.

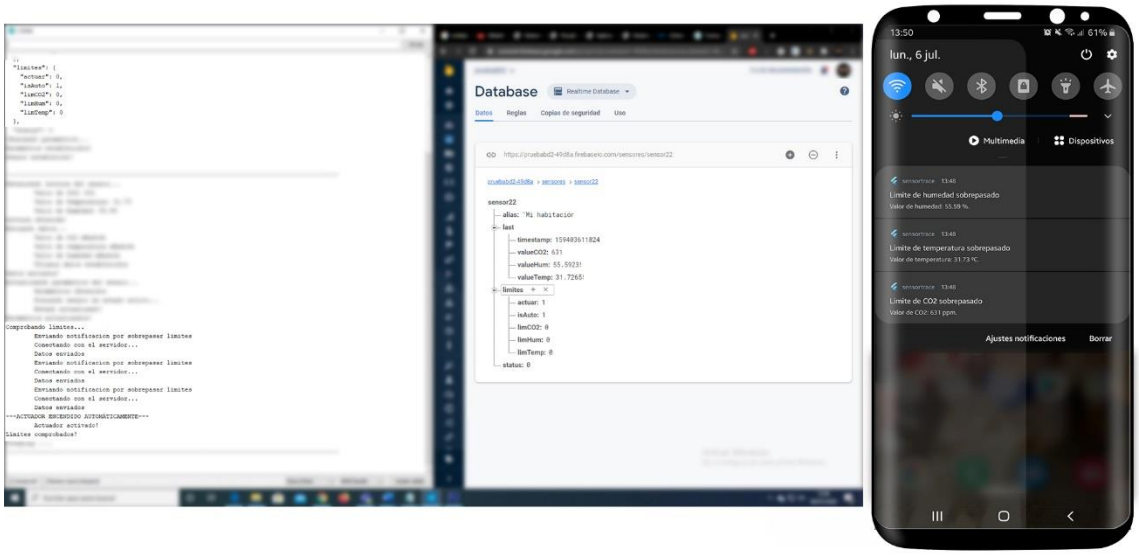


Figura 27. Prueba de envío de notificaciones.

Como se puede ver en la

Figura 27, los últimos valores establecidos, visibles en la parte del centro, superan los límites. Cuando el nodo lo comprueba, envía las notificaciones correspondientes, visible en la parte izquierda. En la parte derecha de la figura, se aprecia como las notificaciones llegan al dispositivo.

En cuarto lugar, se ha probado el funcionamiento del actuador, tanto en modo automático como en manual, para comprobar que funciona correctamente. Todas las figuras que vienen a continuación están compuestas por la consola de *Arduino IDE* a la izquierda y la consola de la base de datos de *Firebase* a la derecha.

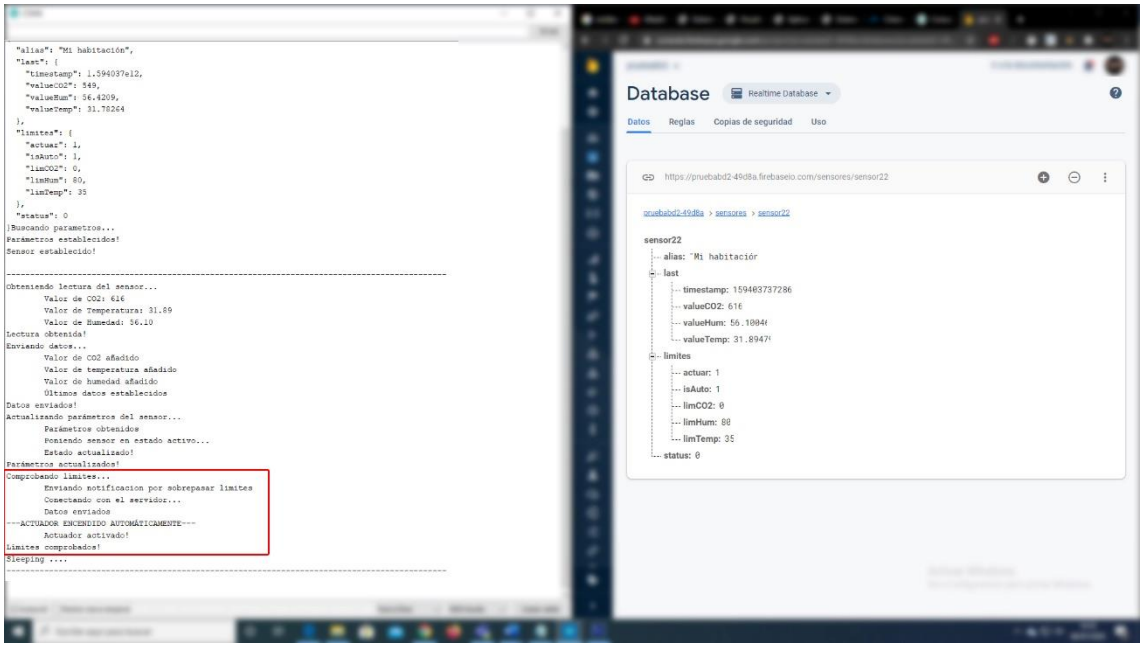


Figura 28. Prueba de activación del actuador en modo automático.

En la Figura 28 se puede observar que el nodo, en modo automático, activa el actuador cuando detecta que alguno de los límites, en este caso CO_2, ha sido superado. En el recuadro rojo se contempla lo que el nodo realiza, mientras que en la consola de *Firebase* se puede ver el estado del nodo.

La propiedad *actuar* puesta a uno indica que el actuador se ha activado o se tiene que activar, *isAuto* establecido a uno indica que el modo de funcionamiento es automático, *valueCO2* representa el último dato obtenido por el nodo y *limCO2* representa el límite establecido.

Observando dichas propiedades, se puede ver que el último valor de CO_2 ha superado su límite y, como el modo de funcionamiento es automático, el propio nodo ha activado el actuador.

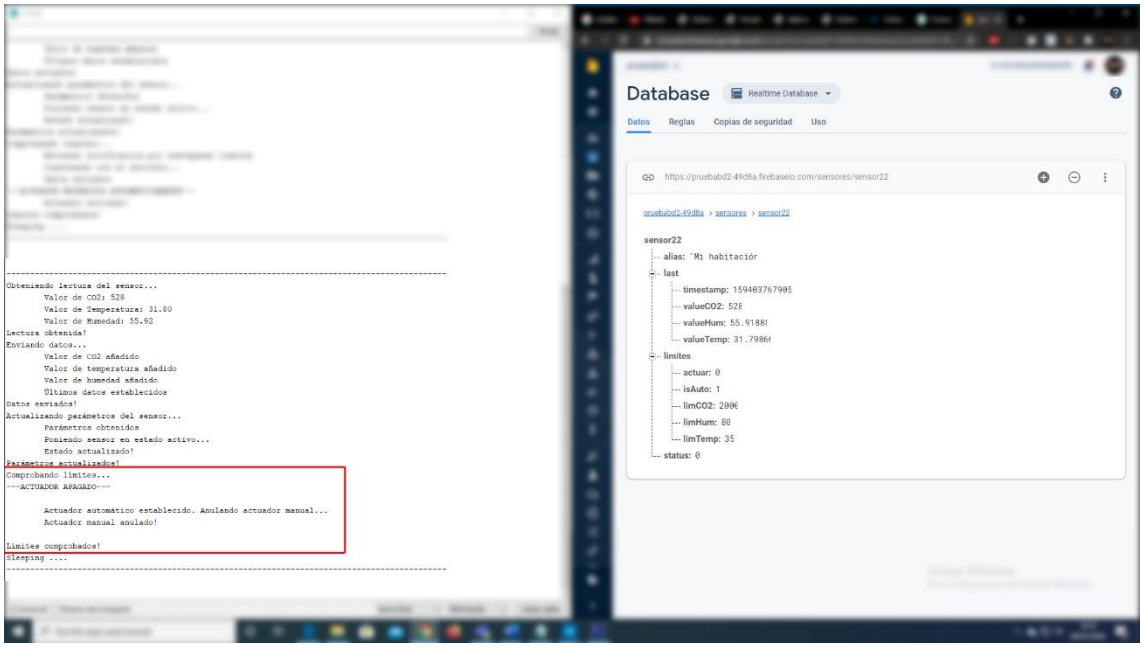


Figura 29. Prueba de desactivación del actuador en modo automático.

Siguiendo el mismo procedimiento, en la Figura 29 se puede comprobar que, cuando ninguno de los valores supera sus límites y el modo de funcionamiento es automático, el nodo desactiva el actuador.

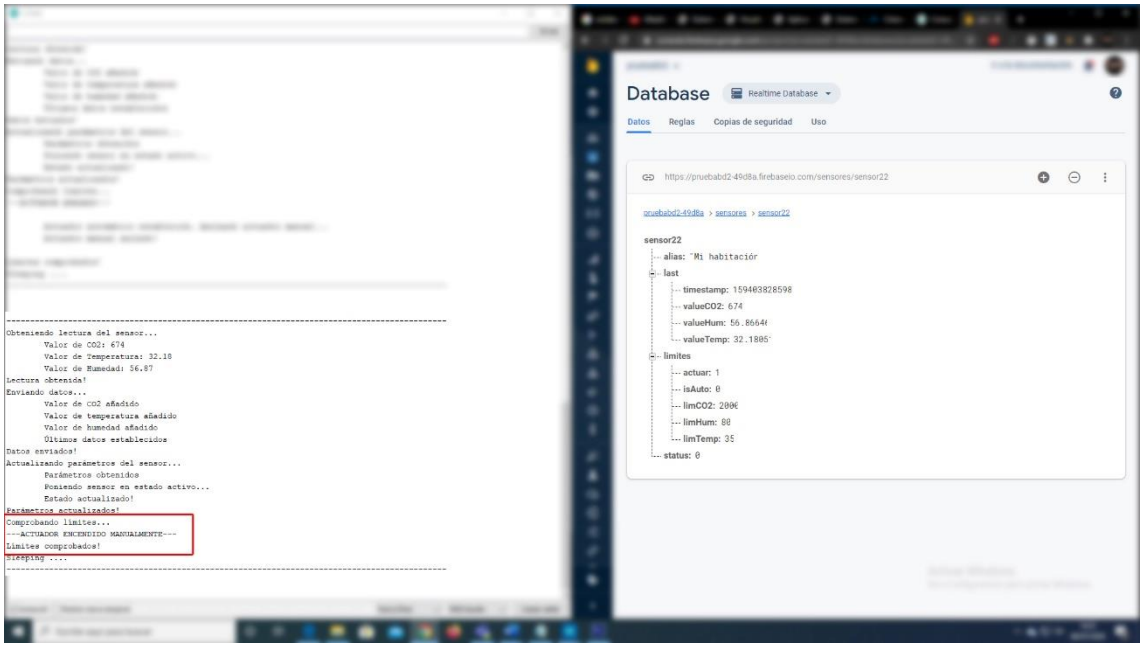


Figura 30. Prueba de activación del actuador en modo manual.

En la Figura 30 se cambia el modo de funcionamiento a manual, es decir, la propiedad *isAuto* se establece en cero. Observando dicha figura, se puede comprobar que el actuador se enciende manualmente a pesar de que ninguno de los últimos valores se ha visto superado.

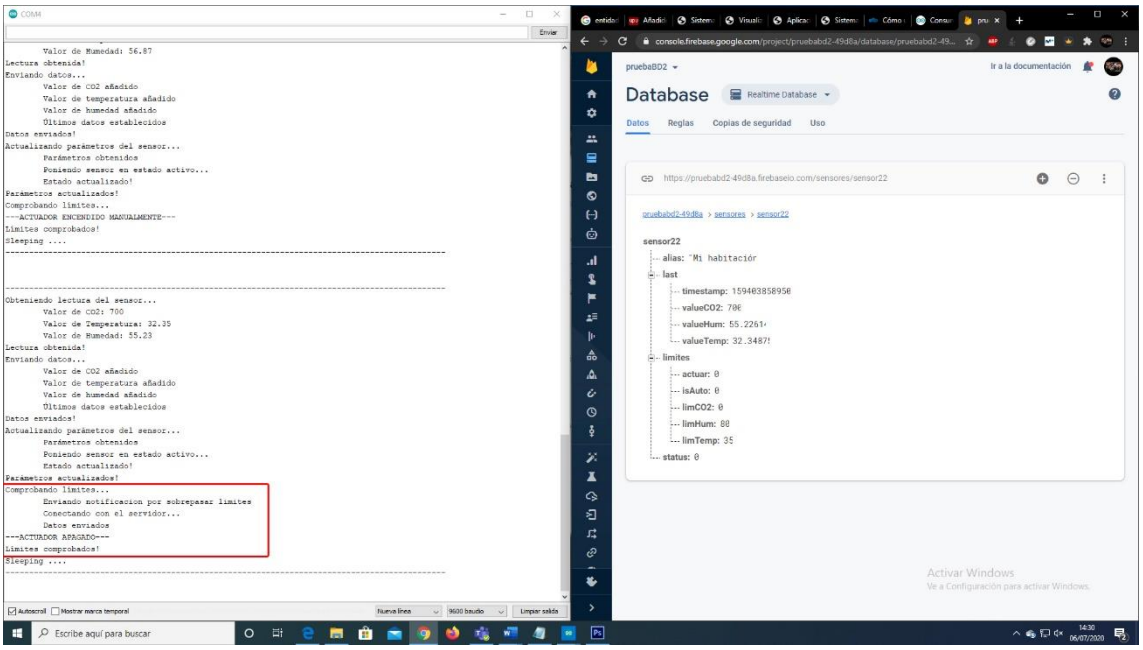


Figura 31. Prueba de desactivación del actuador en modo automático.

En la Figura 31 se encuentra el caso contrario. Observando la figura, se puede comprobar que esta vez el límite de CO_2 se rebasa, pero, al estar en modo manual, el nodo no activa el actuador porque la propiedad *actuar* está fijada en cero.

Para finalizar, se ha hecho una prueba de integración para comprobar el correcto funcionamiento de todo el sistema, cuando el nodo aún no existe, así como cuando el nodo ya existe.

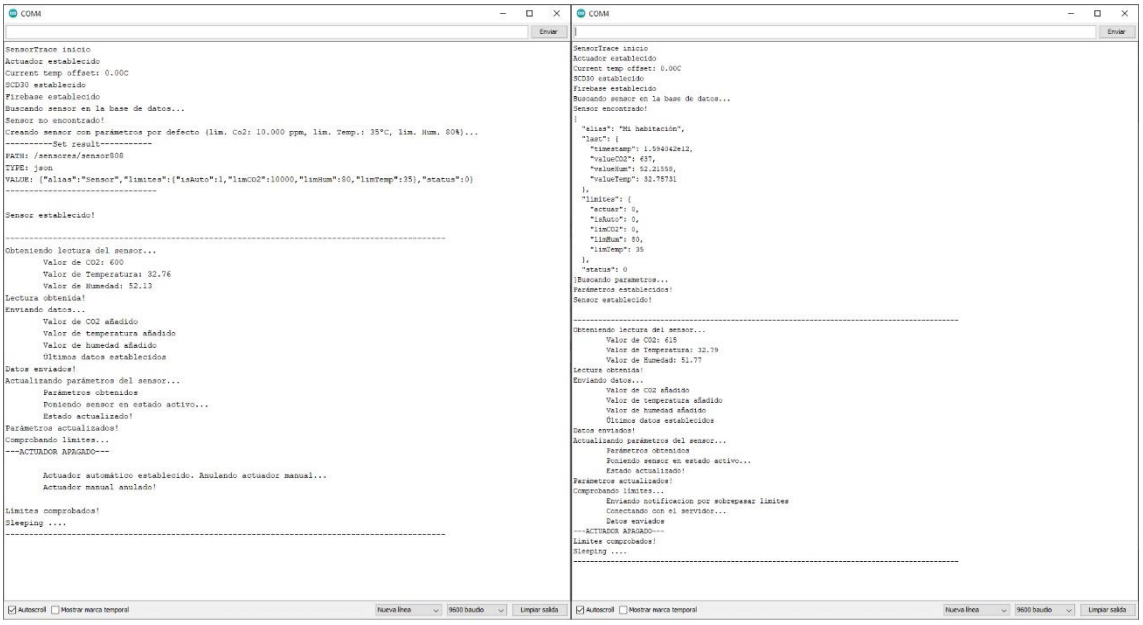


Figura 32. Prueba de integración del nodo.

Se puede observar en la parte izquierda de la Figura 32 que el nodo no existe en la base de datos y por tanto se añade a él mismo. En la parte derecha de la misma figura, se puede ver que el nodo ya existe en la base de datos y captura su estado.

Una vez completada esa tarea, el nodo sigue si funcionamiento en bucle sin ningún problema.

7.2 Aplicación móvil

Para realizar las pruebas en la aplicación móvil, se han seguido los mismos principios que en el apartado anterior. De esta manera, se ha comprobado que cada una de las vistas que conforman la aplicación funcionan correctamente. Además, se ha realizado una prueba de rendimiento para cada una de las vistas. Dicha prueba se ha analizado en un dispositivo *Samsung S9+*, con la versión de *Android 10*. Los resultados de las pruebas se muestran en la siguiente tabla.

	Imágenes por segundo	Uso de memoria (MB)
Vista inicial	60.0	18.7
Vista gráfica	57.6	14.6
Vista lista	58.6	13.8
Vista ajustes	60.0	14.5

8. Conclusiones

Las conclusiones obtenidas tras realizar el proyecto se basan en tres conceptos.

El primer concepto está relacionado con el objetivo de desarrollar el sistema embebido formado por un microcontrolador y un sensor capaz de obtener datos en términos de calidad de aire.

El desarrollo de un programa para tratar con la placa *Arduino* y el sensor *SCD30* suponía uno de los retos más complicados, ya que la experiencia dentro del sector de los microcontroladores y el uso del lenguaje de programación *C* era escasa. Sin embargo, la arquitectura propuesta para la solución de este programa simplificó mucho las cosas al permitir que se aislaran cada una de las funciones. De esta manera, si alguno de los componentes fallaba, se podía identificar y arreglar fácilmente.

Uno de los problemas que más esfuerzo supuso fue tratar con el servicio de mensajería desde la placa. En un primer momento y teniendo en cuenta que la comunicación sería a través de un servicio *REST*, parecía algo simple, pero a la hora de la verdad no fue así.

Para llevar a cabo dicha funcionalidad, se necesitó estudiar detenidamente tanto la estructura que requerían las peticiones como la obtención de la clave secreta del servidor y el *TOKEN_ID* del dispositivo al que se quería enviar las notificaciones.

De hecho, este problema supuso uno más grande, ya que, las notificaciones deberían haberse enviado desde la propia aplicación, añadiendo así la posibilidad de desactivarlas, cosa que no se puede ofrecer si es el nodo el que envía las peticiones.

Aun así, el desarrollo de la placa y el sensor ha concluido con éxito.

El segundo concepto está relacionado con el objetivo de configurar y desarrollar una plataforma *IoT* como base de comunicación entre el nodo y la aplicación móvil.

El proceso de configurar y dar forma a *Firebase* para que sea el soporte entre el nodo y la aplicación ha sido el más simple pero el más largo. A lo largo del proyecto, se han hecho multitud de modificaciones en la estructura de la base de datos hasta llegar a la estructura final. Además, conforme se iba desarrollando el nodo y la aplicación, se requería de nuevas configuraciones y de obtener nuevos parámetros de los servicios para poder avanzar.

Una de las cosas que más impresionan a la hora de estudiar *Firebase,* es la cantidad de servicios que ofrece y la facilidad que tiene para integrarse casi con cualquier servicio de *Google*, su creador. Un ejemplo claro son el servicio de base de datos que incluye dos modelos y el servicio de mensajería, ambos complementados tanto con una *API* para su uso, así como un servicio *REST*, permitiendo que casi cualquier dispositivo con internet pueda usarlos.

No cabe duda de que *Firebase* ha sido la mejor o una de las mejores opciones para este proyecto por la cantidad y calidad de servicios que ofrece, además de ofrecer una gran compatibilidad con *Flutter*, también creada por *Google* y usada para el desarrollo de la aplicación de este proyecto. Por tanto, se puede afirmar que se ha llevado a cabo con éxito la configuración y el desarrollo de la plataforma *IoT*.

El tercer y último concepto está relacionado con el desarrollo de la aplicación móvil para monitorizar los diferentes nodos y todos sus datos.

Desarrollar la aplicación móvil ha sido la parte más difícil debido a que la experiencia con las herramientas de *Flutter* y el lenguaje de programación *Dart* era nula. Para poder desarrollar la aplicación, primero se ha tenido que estudiar detenidamente la arquitectura de *Flutter* basada en *Widgets* y su funcionamiento a través de *Dart*.

Una vez comprendido, se han llevado a cabo una gran cantidad de experimentos en busca de la solución presentada en el proyecto. Estos experimentos han incluido diferentes vistas con sus respectivas ventajas y desventajas, así como diferentes librerías que finalmente no se han incluido. Un ejemplo de ello es el gráfico mostrado en la Figura 15, donde se ha experimentado con dos librerías distintas. Cada una de las librerías ha sido estudiada y probada a fondo para ver cual obtiene mejores resultados tanto visuales como de rendimiento.

Tras todos esos experimentos se ha obtenido un resultado muy bueno, donde las vistas quedan bien definidas e incluyen gran funcionalidad, sin restar peso al rendimiento. Por tanto, el tercer objetivo queda concluido con éxito.

En definitiva, los tres objetivos propuestos para el proyecto se han conseguido con éxito obteniendo así un sistema completo capaz de analizar el aire de nuestro entorno y ayudándonos a conocerlo un poco más en cuanto a calidad de aire.

8.1 Relación del trabajo desarrollado con los estudios cursados

Todas las asignaturas cursadas durante la carrera han aportado su grano de arena. Sin embargo, algunas de ellas han incidido con un mayor grado en el presente proyecto. A continuación, se destacan.

- La asignatura *Gestión de proyectos* ha sido clave para abordar el presente proyecto gracias a las técnicas y métodos enseñadas.
- Las asignaturas *Tecnología de Sistemas de Información en la Red* y *Diseño y aplicaciones de los sistemas distribuidos* han aportado gran cantidad de conocimientos acerca de cómo plantear y diseñar el sistema completo formado por el nodo, la plataforma *IoT* y la aplicación móvil.
- Las asignaturas *Estructura de computadores* y *Sistemas empotrados y de tiempo Real* han sido fundamentales tanto para entender las arquitecturas y el funcionamiento de las placas y los sensores estudiados como para su diseño dentro del proyecto.
- Las asignaturas *Fundamentos de Sistemas Operativos* y *Diseño de Sistemas Operativos* han introducido el lenguaje de programación *C* necesario para poder programar la placa y el sensor.
- Las asignaturas *Redes de Computadores* y *Tecnología de Redes* han aportado conceptos esenciales para entender y diseñar el modelo de comunicación del proyecto.
- La asignatura *Base de datos y sistemas de información* ha aportado las nociones necesarias para un diseño efectivo y de calidad en la base de datos de este proyecto.
- La asignatura *Ingeniería del software* ha aportado los conocimientos necesarios para plantear la arquitectura de la aplicación móvil y realizar un diseño eficiente y de calidad.

- La asignatura *Interfaces Persona Computador* ha permitido realizar el diseño y la programación de las diferentes vistas de la aplicación móvil gracias a los conceptos enseñados.
- Las asignaturas *Lenguajes, Tecnologías y Paradigmas de la Programación* y *Estructura de datos y algoritmos* han sido esenciales en la programación de la aplicación móvil aportando conceptos básicos y complejos para poder tratar con toda la información
- Además de los conocimientos teóricos que se han impartido durante la carrera, otro de los conceptos clave ha sido la inclusión y puesta en práctica de las competencias transversales y específicas de cada rama. En el presente proyecto se han necesitado las siguientes competencias.
- *CT1 – Comprensión e integración* en un grado excelente, para poder explicar de forma clara y concisa los objetivos del proyecto, su desarrollo y los resultados.
- *CT2 – Aplicación y pensamiento práctico* en un grado superior, para poder plantear el estudio y recopilar información de las diferentes posibilidades tecnológicas para llevar a cabo el proyecto.
- *CT3 – Análisis y resolución de problemas* en un grado superior, para poder abordar el problema planteado, así como las decisiones necesarias para establecer los requisitos de cada componente del proyecto.
- *CT5 – Diseño y proyecto* en un grado superior, para la toma de decisiones en cuanto a la solución del problema y su desarrollo durante el proyecto.
- *CT7 – Responsabilidad ética, medioambiental y profesional* en un grado medio, para valorar el trabajo ajeno incluido en este proyecto.
- *CT8 – Comunicación efectiva* en un grado superior, para atender las pautas marcadas en el desarrollo de la memoria del proyecto y usar un léxico adecuado respetando las normas lingüísticas.
- *CT9 – Pensamiento crítico* en un grado superior, para poder opinar y tomar decisiones acerca de las tecnologías utilizadas para el desarrollo del proyecto.
- *CT10 – Problemas contemporáneos y ODS* en un grado medio, para establecer el impacto que supone el presente proyecto.
- *CT11 – Aprendizaje permanente* en un grado superior, para poder enfrentar los diferentes problemas encontrados a lo largo del desarrollo del proyecto.
- *CT13 – Instrumentación específica* en un grado medio, para poder defender las diferentes tecnologías utilizadas en el proyecto.
- *EC2 – Capacidad de desarrollar procesadores específicos y sistemas empotrados, así como desarrollar y optimizar el software de dichos sistemas* en

un grado superior, para desarrollar el software del sistema empotrado presente en el proyecto.

- *EC4 – Capacidad de diseñar e implementar software de sistema y de comunicaciones* en un grado superior, para el desarrollo de todos los componentes del sistema, incluyendo la placa con el sensor, la plataforma *IoT* y la aplicación móvil.
- *EC5 – Capacidad de analizar, evaluar y seleccionar las plataformas hardware y software más adecuadas para el soporte de aplicaciones empotradas y de tiempo real* en un grado superior, para poder seleccionar los componentes del sistema empotrado y la plataforma *IoT* usadas en el proyecto.
- *EC7 – Capacidad para analizar, evaluar, seleccionar y configurar plataformas hardware para el desarrollo y ejecución de aplicaciones y servicios informáticos* en un grado medio, para configurar las herramientas necesarias para el desarrollo del sistema empotrado y la aplicación móvil.

9. Trabajos futuros

El principal fleco de este proyecto han sido las pruebas. Es verdad que se han comprobado todos los componentes que forman tanto al nodo como a la aplicación móvil y tras muchas modificaciones a lo largo del proyecto, han salido exitosas. Sin embargo, debido al tiempo y a la falta de instrumental, hay otras muchas pruebas que no se han llevado a cabo. Ejemplo de esto se puede encontrar en las pruebas no funcionales de los componentes, excluyendo la prueba de rendimiento de la aplicación.

Queda, por tanto, como trabajo futuro, llevar a cabo pruebas no funcionales tanto en el nodo como en la aplicación. Dichas pruebas incluyen el consumo, el rendimiento y pruebas de seguridad que no se han considerado de momento, entre otras.

Bibliografía

[1] P. Giner, C. Cetina, J. Fons y V. Pelechano, «Developing Mobile Workflow Support in the Internet of,» València, 2010.

[2] R. Loureido Garrido, «Estudio Plataformas IoT,» 2015.

[3] A. Reyes Díez, «Integración de dispositivos IoT en una red comunitaria,» País Vasco, 2017.

[4] R. Ordóñez Rodríguez, «Sistemas de monitorización basado en plataformas arduino, android y cloud,» Málaga, 2015.

[5] D. Rodríguez González, «Arquitectura y gestión de la IoT,» vol. 12, nº 3, p. 12, 2013.

[6] M. R. Pérez, M. A. Mendoza y M. J. Suárez, «Paradigma IoT: desde su conceptualización hacia su aplicación en la agricultura,» *Espacios,* vol. 40, nº 18, p. 6, 03 06 2019.

[7] E. A. Quiroga Montoya, S. F. Jaramillo Colorado, W. Y. Campo Muñoz y G. E. Chanchí Golondrino, «Propuesta de una arquitectura para agricultura de precisión soportada en IoT,» *Revista Ibérica de Sistemas e Tecnologias de Informação,* pp. 39-56, 2017.

[8] R. Enríquez Herrador, «Guía usuario arduino,» 2009.

[9] «Arduino nano,» 2018.

[10] «Infrared CO2 sensor module,» 2019.

[11] «Datasheet sensirion SCD30 sensor module,» 2020.

[12] E. F. Campelo Rivadulla, «Evaluación de servicios en la nube para el desarrollo de aplicaciones IoT sobre redes inalámbricas de sensores,» 2017.

[13] V. Fernández y J. Leyton, «Cloud computing,» 2010.

[14] C. Rodríguez y H. Enríquez, «Características del desarrollo en Frameworks multiplataforma para móviles,» *Ingenium,* vol. 15, nº 30, pp. 101-117, 10 2014.

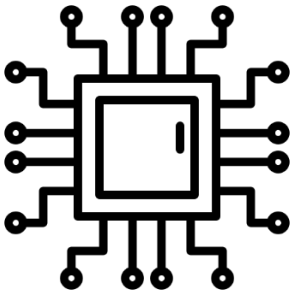
[15] J. D. Serrano Andrés, «Diseño, implementación e integración de un sistema de medición de variables de entorno en un sistema IoT con Software y Hardware libre,» 2017.

[16] E. González Daza, «Red de sensores-Internet de las cosas,» 2015.

Anexos.

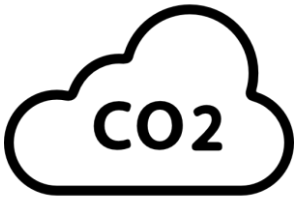
Anexo1.

Imagen circuito.



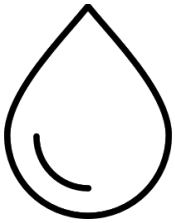
Anexo 2.

Imagen C02.



Anexo 3.

Gota de agua.



Anexo 4.

Termómetro.