UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Superior d'Enginyeries Industrial, Aeroespacial i Audiovisual de Terrassa

# Estudio de un sistema IoT para su aplicación en gestión de Smart Cities

TREBALL FINAL D'ESTUDIS

Documento:

Memoria

Autor:

Callejón Rodríguez, Rafael

Director /Co-director:

Ortega Redondo, Juan Antonio
/ Romeral Martínez, José Luis

Titulación:

Grado en Ingeniería en Tecnologías Industriales

Convocatoria:

Prórroga (2021 – 2022)

## Resumen

Los sistemas basados en IoT cada vez son más presentes tanto en la vida cotidiana como en entornos empresariales e industriales, gracias a los últimos avances tecnológicos y a la dirección que está tomando la sociedad. Cada vez son más comunes los procesos automáticos, inalámbricos y que requieran de una transmisión de datos sin intervención humana máquina a máquina.

En este proyecto se realizará un desarrollo de un sistema basado en IoT que permita monitorizar, estudiar y gestionar datos recopilados mediante nodos, de forma remota, encargados de recopilar datos medioambientales.

Para llevar a cabo el proyecto, se ha propuesto que uno de los nodos sea un nodo físico en base a la placa de desarrollo Arduino UNO y el resto de nodos, que sean nodos creados a partir de bases de datos públicas.

Como protocolo de comunicación red se ha utilizado el bróker MQTT, que nos ofrece la capacidad de comunicar los dispositivos mediante publicaciones y suscripciones.

Cuanto a la gestión de los nodos remotos, se realizará mediante un entorno de programación visual llamada Node-Red, donde podremos establecer relaciones entre nuestros dispositivos de hardware (mediante el bróker MQTT), web API y bases de datos.

Los datos obtenidos se irán publicando y almacenando en la base de datos NoSQL de Firebase Real-Time Database, alojada la nube.

Para monitorizar los datos mediante una interfaz gráfica de usuario, aprovechando la potencia de Firebase Real-Time Database, se creará una aplicación móvil que visualice los datos en tiempo real diseñada desde cero.

## Abstract

IoT-based systems are becoming more present both in everyday life and in business and industrial environments, due to the latest technological advances and the direction that society is taking. Automatic, wireless processes that require data transmission without human intervention machine-to-machine are becoming more common.

This project will develop a system based on IoT that allows monitoring, studying and managing data collected through wireless remote nodes in charge of collecting environmental data.

To carry out the project, it has been proposed that one of the nodes has to be a physical node based on the Arduino UNO development board and the rest of the nodes are nodes generated from public databases.

MQTT broker has been chosen as the transmission protocol, which offers us the ability to communicate between devices through publications and subscriptions.

For the management of remote nodes, a visual programming tool called Node-Red will be ensured, where we will be able to establish relationships between our hardware devices (through the MQTT broker), web API and databases.

The data obtained will be published and stored in the Firebase Real-Time Database NoSQL database, hosted in the cloud.

To monitor the data through a graphical user interface, taking advantage of the power of the Firebase Real-Time Database, a mobile application will be created that visualizes the data in real time designed from scratch.

# Índice

# Índice de tablas

# Índice de figuras

## Lista de abreviaturas

IoT: Internet of Things, Internet de las Cosas.

M2M: Machine-to-machine, Máquina a máquina.

SoC: System on a Chip.

Npm: Sistema de gestión de paquetes en Node.js.

AVR: Familia de controladores RISC, desarrollados y fabricados por Atmel.

RISC: Reduced Instruction Set Computing. Diseño de CPU enfocado a un conjunto de instrucciones pequeñas.

CPU: Central Processing Unit. Componente encargado de dirigir y ejecutar las tareas que lleva a cabo un dispositivo.

MQTT: TQ Telemetry Transport. En telecomunicación, pertenece al sistema de transmisión de datos mediante publicadores y suscriptores basado en un bróker y clientes.

CoAP: Constrained Application Protocol.

LoRaWAN: Low Power Wide Area Network.

CDA: Calidad de aire

BDD: Base de datos.

TCP/IP: Transfer Control Protocol. Protocolo a través del cual se realiza una transmisión de datos entre los participantes de una red.

## Glosario

Topic: Refiriéndonos al protocolo MQTT, es el tema al que irá referido el mensaje a publicar y al cuál se suscribirá el cliente.

Payload: Referido al mensaje (datos) que se envía al topic.

# 1. Introducción

## 1.1 Objetivo

**Objetivo general**

• Desarrollar una plataforma IoT en la que se puedan visualizar y gestionar datos obtenidos mediante una red de nodos y transmitidos a través de un protocolo IoT.

**Objetivos específicos**

• Evaluar distintas alternativas para la generación de los nodos, protocolos de transmisión de información, bases de datos e interfaz de usuario.

• Realizar el desarrollo físico de uno de los nodos.

• Utilizar un protocolo de transmisión IoT para el envío de datos de los nodos.

• Desarrollar una interfaz gráfica de usuario en una aplicación móvil.

• Validar el correcto funcionamiento del sistema realizado.

## 1.2 Alcance

**Red de nodos**

• La red de nodos en este proyecto está formada por un único nodo físico y el resto son generados a partir de información obtenida de bases de datos públicas.

• El nodo físico es capaz de recolectar información de temperatura, humedad y calidad de aire. Los nodos generados a partir de información de fuentes de datos públicas, obtendrán medidas de temperatura y humedad.

• El protocolo de red utilizado en el nodo físico es WiFi, para establecer la conexión con la red inalámbrica.

• La transmisión de datos para el nodo físico es unidireccional, es decir, el flujo de información irá desde el nodo hacia la nube.

• Los nodos realizan mediciones pertenecientes a distintas zonas geográficas:

- Nodos remotos: Badalona, Barcelona, Granollers y Terrassa.
- Nodo físico: Martorelles.

**Red de comunicaciones y la Nube (Cloud)**

• Los datos recopilados por el nodo físico son transmitidos mediante un protocolo IoT. Posteriormente, son procesados para ser almacenados en una base de datos en la nube.

• Los datos extraídos de una base de datos pública, son procesados para ser almacenados en la base de datos en la nube.

**Interfaz de usuario**

• La interfaz de usuario es una aplicación desarrollada para dispositivos móviles Android.

• La aplicación está vinculada a la base de datos en la nube, de la cual se extraen los datos.

• En cuanto a la interacción con el usuario, se han desarrollado varias herramientas: filtros de búsqueda entre los nodos, gráficos, listados de mediciones y pantallas con subprogramas de monitorización.

## 1.3 Requerimientos

• En primera instancia, se requirió que el desarrollo del nodo físico fuera en base a la placa de desarrollo Arduino UNO/Genuino UNO (microcontrolador ATmega328P).

• Generar una red de nodos pese a realizar únicamente el desarrollo de uno de los nodos en cuanto a arquitectura física, deberán generarse nodos obteniendo la información aportada por bases de datos públicas.

• Utilizar un protocolo de IoT para la transmisión de datos a través de los nodos.

• Desarrollar una interfaz de usuario que permita la visualización de los datos obtenidos mediante la red de nodos.

## 1.4 Justificación

Desde un punto de vista personal, como alumno, con interés y curiosidad en la programación y desarrollo, buscaba un acercamiento algo más próximo a entender el funcionamiento de hardware como son los microcontroladores, sensores y módulos auxiliares a el interés por programar en distintos ámbitos, desde programar hardware hasta bases de datos, programación web y aplicaciones y utilizar nuevos entornos de desarrollo.

Partiendo de un punto de vista global, el proyecto pretende ofrecer una solución simple, económica y portable, para la transmisión de datos mediante una red de nodos y ser capaces de almacenar los datos obtenidos para su posterior gestión. Gracias a que el protocolo seleccionado para el desarrollo es un protocolo IoT, los nodos que conforman la red podrán ser remotos, estableciendo una red de conexión inalámbrica.

Por lo que se trata de un proyecto educativo y formativo, además de realizar un proyecto funcional que puede ser útil para sistemas IoT de control muy diversos, como en este caso: meteorológico y atmosférico por zonas geográficas, ciudades o poblaciones. Este sistema nos puede servir tanto para obtención de datos en interiores de casas aplicables en casos tales como control de la calefacción, suministros de agua, control de la situación de una vivienda mediante avisos… Como exteriores, por ejemplo, localizar puntos de alto o bajo consumo energético, regular el tráfico para disminuir el impacto de las emisiones de $CO_2$ en base a la información recopilada en los nodos...

En conclusión, se trata un sistema aplicable a distintos casos, dependiendo de las prestaciones y periféricos que se decida aplicar a cada nodo.

# 2 Antecedentes

Un sistema IoT consiste en conectar los objetos que nos rodean en nuestra vida cotidiana de forma digital entre sí, de manera inalámbrica y mediante Internet. El IoT pretende que las personas no tengan que intervenir en estas conexiones entre objetos, es decir, que las conexiones sean M2M (máquina a máquina).

## 2.1 Plataformas IoT

"Las plataformas IoT son el software que permite conectar dispositivos, sensores, actuadores y equipos industriales en un entorno digital, generando una red para que éstos puedan comunicarse y crear información valiosa. " (Barbara IoT, 16 de abril de 2021). Los elementos que componen la plataforma son:

**Hardware**. Son los dispositivos conectados. Para el caso de este proyecto, la parte perteneciente al hardware es la del desarrollo de los nodos.

**Conectividad**. Es la puerta de enlace a Internet. Incluye los protocolos de transmisión y de red que permite el transporte de datos desde el nodo físico hacia el software.

**Software**. En una plataforma IoT, se suele referir al código en la nube, que conforman la gestión de los datos.

**Interfaz de acceso**. Es el punto de acceso del usuario con el sistema IoT.

En la metodología y estructuración del trabajo, se ha juntado la conectividad y el software en un solo bloque, llamado red de comunicaciones y la Nube.

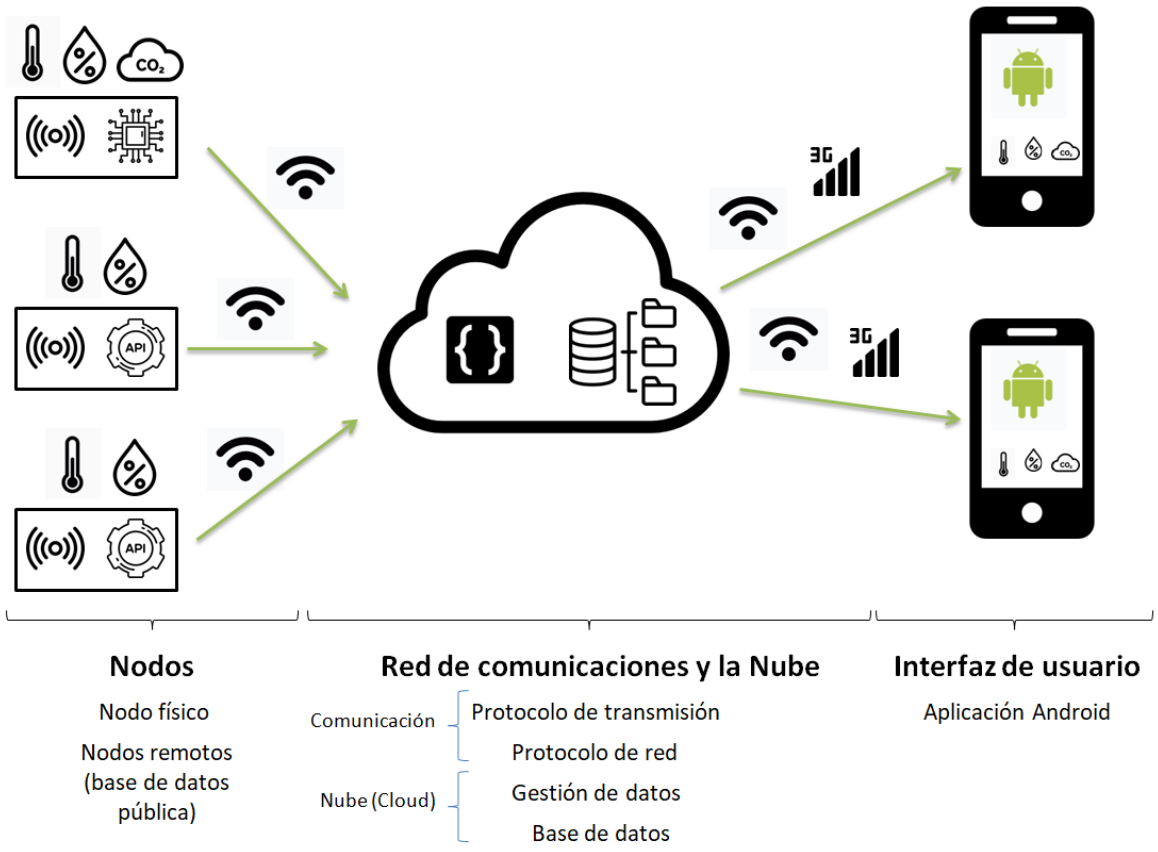


*Figura 1:* Rafael C. (2022) [Estructura del proyecto]. Fuentes: Iconos obtenidos de https://www.flaticon.es/

## 2.2 Smart City

En el campo de las ciudades inteligentes (Smart Cities), los sistemas IoT se entienden como una infraestructura de comunicación capaz de recopilar información procedente de una red de sensores distribuida, capaz de interpretar los datos de los sensores, reformatearlos y realizar una toma de decisiones en cuanto los resultados.

La idea de Smart City nació debido al incremento de población mundial, junto con el incremento de concentración poblacional en grandes centros urbanos, de forma que las ciudades deben buscar soluciones para prevenir el colapso y garantizar la sostenibilidad. Esto es debido a que los modelos con los que se diseñaron las ciudades quedaron obsoletos debido a las altas exigencias energéticas, emisiones de $CO_2$, incremento del tráfico, provisión de bienes y materias primas y la prestación de servicios sanitarios.

El modelo ideal de una Smart City posee diversos subsistemas: una generación energética distribuida y repartida por el territorio, redes inteligentes de datos bidireccionales entre usuario y centro de control, un sistema de medición energético inteligente de cada usuario a través de telecontadores, edificios inteligentes con prestaciones de domótica que respetan el medio ambiente, una red de sensores inteligentes que recopilan datos necesarios encargados de que cada subsistema siga cumpliendo su función, implantación de vehículo eléctricos y distintos puntos de carga, establecer tecnologías TIC y que la sociedad esté involucrada en la Smart City.

## 2.3 Tecnologías disponibles

Debido a la creciente popularidad que está teniendo el IoT y la revolución que está suponiendo la Industria 4.0, el sector está sufriendo una evolución acelerada, por lo que cada vez son más numerosas la variedad de opciones y herramientas que salen al mercado para este tipo de desarrollos.

### 2.3.1 Hardware. Arquitectura del nodo

A fecha de la realización de este documento, para la generación de nodos en desarrollos de IoT, las placas de desarrollo más extendidas debido a la cantidad de aplicaciones IoT en las que están presentes son las placas de Arduino, Raspberry o NodeMCU.

**Arduino** a día de hoy se ha convertido en uno de los referentes y estándares para el desarrollo de hardware, gracias a la enorme cantidad de módulos, extensiones, librerías y soporte por parte de la industria. Debido a la flexibilidad que ofrecen estas placas para cualquier tipo de proyecto o aplicación y precio razonable, Arduino es una de las mejores opciones por las que empezar el desarrollo.

En cuanto a placas de base **Raspberry**, son muy populares debido a que actúa como un computador en miniatura, lo cual ofrece mucha versatilidad para realizar diversas funciones como nodo. A día de hoy, el precio de las series Pi ha cuadruplicado el precio original debido a la crisis de microchips, llegando a superar los 150 € para placas de series más antiguas. Se han lanzado versiones más económicas con menor capacidad de procesamiento, como son las series Pi Zero.

Las placas **NodeMCU** son comúnmente contempladas en desarrollos y proyectos IoT, dado que estas están basadas en el SoC ESP8266, incorporando así un módulo WiFi en el propio kit de desarrollo. Siendo una de las opciones más económicas enfocadas a desarrollos IoT, lista para trabajar mediante un protocolo de red y la compatibilidad que posee con Arduino, posee un uso muy extendido.

### 2.3.2 Red de comunicaciones y la Nube

Esta parte de la tecnología está compuesta por servidores desplegados de forma local o en la propia Nube y los servicios alojados en la Nube.

Los servidores contienen los protocolos de transmisión para ofrecer una puerta de entrada a los datos comunicados por parte de los nodos hacia la Nube o transmitir datos entre servicios dentro de la propia Nube.

#### 2.3.2.1 Protocolos de red

Las alternativas que existen en cuanto a protocolos de acceso a red de forma inalámbrica son las redes **WiFi**, **3G**, **4G** o **5G**.

#### 2.3.2.2 Protocolos de transmisión

Los protocolos de transmisión IoT más extendidos a día de hoy son MQTT, CoAP, LoRa y HTTP.

- El protocolo **MQTT** es un protocolo M2M de mensajería basado en la pila TCP/IP (por lo que las conexiones que se realicen mediante este protocolo se mantendrán y reutilizarán). El funcionamiento consiste en hacer de bróker entre dispositivos, mediante un sistema de publicadores y suscriptores a un topic (tema) en concreto. De esta forma, los dispositivos conectados y suscritos al topic, recibirán el mensaje publicado en el bróker por los dispositivos conectados y publicadores a través del topic. Gracias a la fácil implementación y apoyo por parte de la mayoría de las plataformas, este protocolo es uno de los más populares y utilizados en IoT.
- El protocolo **CoAP** es un protocolo cliente-servidor que permite enviar paquetes entre nodos, estos paquetes son evaluados por el servidor y, en función de la lógica del paquete, decidirá qué acción realizar.

- El protocolo **LoRaWAN** es un protocolo de largo alcance M2M de bajo consumo, que admite millones de dispositivos conectados y requiere de un operador de telecomunicaciones.

#### 2.3.2.3 Procesamiento y gestión de datos

Para el procesamiento y la gestión de datos existen distintas alternativas válidas en función del tipo de aplicación que queramos realizar. Estas alternativas son o bien utilizar herramientas IoT, plataformas IoT o una combinación de ambas.
En cuanto a herramientas y protocolos IoT de código abierto para la gestión de dispositivos y datos, los más extendidos para aplicaciones de este estilo son Node-RED, Zetta, OpenRemote, ThingsBoard y Thinger:

- **Node-RED** consiste en una interfaz visual de programación mediante nodos. Utilizando JavaScript como lenguaje de programación, permite la conexión entre dispositivos, web APIs y servicios en la nube, además de incluir los protocolos HTTP y

MQTT de base. Es un entorno ampliable gracias al amplio apoyo que posee, ya que permite la instalación de distintos módulos y nodos que permiten aumentar enormemente sus prestaciones. Cabe mencionar que también posee una interfaz de usuario web programable.

- **Zetta** es una herramienta de IoT centrada en APIs, capaz de ejecutarse en la nube, un ordenador o, incluso, en placas de desarrollo. Utiliza JavaScript como lenguaje de programación y, pese a estar centrada en protocolos HTTP para despliegue de HTTP API, soporta la mayoría de protocolos de transmisión utilizados en los dispositivos.
- **OpenRemote** es una plataforma IoT que ofrece un editor y tablero de control para gestionar, procesar y monitorizar los datos, además de ofrecer una interfaz de usuario web. Soporta los protocolos MQTT y HTTP REST para la conexión de dispositivos.
- **Thingsboard** es una de las plataformas IoT más conocidas. Posee un tablero que permite la gestión tanto de dispositivos como de datos, acceso a una interfaz de usuario propia y ofrece ciertas funciones de personalización. Posee todos los protocolos estándar de IoT y es compatible con la mayoría de servicios en la nube.
- **Thinger** es otra plataforma que permite la conexión de dispositivos, gestión de datos y monitorización en tiempo real a través de su interfaz de usuario. La plataforma es compatible con la mayoría de arquitecturas típicas de dispositivos IoT.

#### 2.3.2.4 Base de datos

En cuanto a la base de datos, existen muchas posibilidades. Como opciones de bases de datos comúnmente utilizadas NoSQL en desarrollos IoT:

- **Firebase Real-Time Database**: es una base de datos NoSQL alojada en la nube, cuya notación requerida para el almacenamiento y gestión de los datos es formato JSON. La sincronización de los datos, como su nombre indica, es en tiempo real para todos los clientes conectados a la base de datos, además de permitir el acceso a los datos sin conexión a internet, permitiendo la sincronización automática con los datos más recientes al recuperar la conexión. El acceso a los datos para realizar operaciones CRUD (Crear, Leer, Actualizar y Borrar), permite realizarse o bien mediante la API de Real-Time Database o desde aplicaciones cliente. Posee compatibilidad para lenguajes de desarrollo de aplicaciones multiplataforma y web. El entorno de Firebase también posee funcionalidades, librerías y reglas programables de seguridad propias tanto para la base de datos como para las aplicaciones a desarrollar.
- **MongoDB**: es una solución interesante, ya que ofrece servicios de movilidad a través de AWS (dominio de DynamoDB), Azure (Microsoft), Google Cloud, ofreciendo también una API admitiendo una estructura de datos similar a la planteada en Real-Time Database.
- **DynamoDB**: es la solución que nos ofrece el dominio de Amazon, esta sin duda es la que ofrece mayor capacidad de escalar al contenido del proyecto (sin tener que crear diversas instancias) de forma tanto vertical como horizontal. También ofrece un servicio con disponibilidad de datos crítica, junto con servicios y herramientas del dominio.

Otra opción sería usar bases de datos SQL, únicamente en el caso de que la red de nodos, todos los nodos sean del mismo tipo y realicen el mismo tipo de medición, ya que si no, se generaría redundancia. Las bases de datos más típicas para este tipo de desarrollos son MySQL, Oracle y PostgreSQL. Cabe mencionar que la mayoría de desarrollos en IoT, suelen utilizar bases de datos NoSQL, dado que los datos que se suelen almacenar, suelen venir de distintos tipos de sensores que realicen distintos tipos de medición (evitar problemas de redundancia).

Finalmente, también cabe la posibilidad de utilizar una de las plataformas de IoT inspiradas en bases de datos NoSQL como son M2MLabs (Apache Cassandra NoSQL) y Thinger.

### 2.3.3 Interfaz de usuario

En cuanto al desarrollo de una interfaz de usuario para un sistema IoT, existen gran variedad de opciones, ya sea generándola desde cero mediante un lenguaje de programación o utilizando la propia interfaz de usuario de alguna de las herramientas o plataformas IoT comentadas anteriormente.

En el caso de querer programar una interfaz de usuario desde cero, habrá que decidir en qué plataforma y sistemas operativos funcionará. Para el caso de interfaces web, los lenguajes más comúnmente usados son HTML, CSS y JavaScript.

Si la interfaz a realizar es para dispositivos móviles, habrá que tener en cuenta el sistema operativo en el que se quiera desarrollar, aunque también hay que valorar los lenguajes de desarrollo de aplicaciones multiplataforma:

- Sistemas operativos **Android**: en caso de querer ejecutar la aplicación en Android, se puede utilizar tanto el propio lenguaje Android, como Kotlin o Java. También es compatible con lenguajes de programación web como los mencionados anteriormente. La ventaja de programar en dispositivos Android es que la mayoría de dispositivos son compatibles con el compilador y, en caso de no tener un dispositivo físico disponible, fácilmente se puede simular una instancia mediante Android Studio.
- Sistemas operativos **iOS**: si el desarrollo es para un dispositivo con sistema operativo iOS, el lenguaje a utilizar será Swift o Objective-C. Habrá que tener en cuenta que, en caso de querer desarrollar únicamente para este tipo de dispositivo, será necesario compilar el código de la aplicación en un dispositivo que posea un sistema operativo compatible con macOS, ya que el compilador XCode posee este requisito.
- **Flutter**: se trata de un kit de programación basado en el lenguaje Dart desarrollado por Google, enfocado a la generación de aplicaciones móviles. Mayormente es usado para la generación de interfaces de usuario para aplicaciones en Android, iOS y Web. Al estar basado en un lenguaje multiplataforma, Flutter posee las mismas ventajas que Android y iOS, ya que puede ejecutarse en los mismos entornos de programación que Android, además de permitir realizar aplicaciones y compilar el código en navegadores web.

# 3 Selección de tecnologías utilizadas

En este capítulo, se han seleccionado las tecnologías a utilizar dentro de las alternativas evaluadas en el apartado *Tecnologías disponibles,* dando una breve enumeración y explicación de los motivos por los que se han seleccionado para el desarrollo de la plataforma.

## 3.1 Nodos

### 3.1.1 Nodo físico

**Placa de desarrollo**

La placa de desarrollo seleccionada para la arquitectura del nodo físico es la placa de desarrollo Arduino UNO o Genuino UNO (ATmega328P), ya que la realización del nodo en base a esta placa es un requisito del proyecto.

Desarrollar el nodo físico en base a esta placa tiene sus ventajas, ya que la placa Arduino UNO es uno de los mejores puntos de partida para este tipo de desarrollos, ya que esta placa ofrece:

- Compatibilidad con la mayoría de módulos y sensores.
- Soporte de software de código abierto.
- Gran variedad de distintas aplicaciones y usos.
- Permite realizar desarrollos escalables.
- Bajo coste (económico).

**Sensores**

Existe una gran variedad de sensores en el mercado. Para el desarrollo del nodo físico, se decidió tomar mediciones de temperatura, humedad y calidad de aire:

- Sensor de **temperatura** y **humedad** (DHT11): Se ha seleccionado este sensor debido a que los sensores de la serie DHT son comúnmente utilizado para desarrollos en los que se requieren tomar mediciones ambientales, mediante un mismo periférico obtenemos tanto la medición de temperatura como la de humedad, el software a desarrollar en la placa de desarrollo está incluido en la librería Adafruit Unified Sensor Driver y el precio es muy económico.
- Sensor de **calidad de aire** (MQ-135): Este sensor ofrece una solución para detectar la calidad de aire en el ambiente o, una vez calibrado, detectar una gama de gases para la que es sensible el sensor. Además de su utilidad, este sensor se ha seleccionado debido a que su lectura es interpretada directamente mediante la lectura de su entrada analógica (pese a que posee librerías dedicadas), es muy robusto, estable y económico.

**Protocolo de red para el nodo físico**

El protocolo de red a utilizar para el transporte de los datos en el nodo físico es mediante **WiFi**. Para incluirlo, debido a que la placa de desarrollo Arduino UNO no incluye un módulo, se le ha implementado el mismo módulo que utilizan las placas NodeMCU, el **ESP8266 ESP-01**. Los motivos por los cuales se ha seleccionado este módulo son:

- El firmware del módulo se sigue actualizando.
- Posee comandos AT para verificar el correcto funcionamiento.
- Soporta diversas librerías compatibles con Arduino UNO.
- Da acceso al uso de protocolos de transmisión IoT inalámbricos (MQTT).

- Es utilizado en un gran número de desarrollos de IoT, IIoT, redes de sensores y Smart Houses.

### 3.1.2 Nodos remotos

Durante la selección de qué base de datos pública utilizar para la obtención de las mediciones, se han probado distintas fuentes. La decisión final a tomar fue entre utilizar la web API proporcionada por la base de datos de AEMET o Visual Crossing Weather. Finalmente, se decidió utilizar como base de datos pública para la generación de mediciones de nodos remotos **Visual Crossing Weather** por los siguientes motivos:

| | **Visual Crossing Weather** | **AEMET** |
|---|---|---|
| **Contenido** | Condiciones actuales<br>Predicción (14 días) | Predicción (5 días) |
| **Notación de los datos** | JSON | XML |
| **Protocolo HTTP** | Sin problemas | Sin problemas |
| **Generar objeto para procesar datos (Node-RED)** | Sin problemas | Problemas (caracteres especiales tipo $ y _) |
| **Filtrado de datos** | Fácilmente (pocas capas) | Difícil (muchas capas y modelo de datos con tablas vacías) |
| **Actualizaciones** | 10 – 30 minutos | Diariamente |

*Tabla 1:* [Comparación entre bases de datos públicas]

## 3.2 Red de comunicaciones y la Nube

### 3.2.1 Comunicación. Protocolo de red

Como se ha mencionado anteriormente en el apartado *"Protocolo de red para el nodo físico",* el protocolo de red seleccionado es **WiFi**, de forma que para dar acceso a la conexión WiFi, se utilizará un router WiFi y, debido a que el despliegue de los servidores en la nube se realizará mediante protocolos WiFi en un ordenador de sobremesa, es necesario utilizar un módem para establecer la conexión inalámbrica al router y permitir así la conexión entre dispositivos de forma inalámbrica.

### 3.2.2 Comunicación. Protocolo de transmisión

Dado que LoRa es un protocolo de transmisión pensado para dar solución a zonas donde el requerimiento de energía y alcance de la red son un problema, para el caso de una Smart City, donde estas causas no son un problema, será más oportuno utilizar el protocolo MQTT o CoAP.

| Características | MQTT | CoAP |
|---|---|---|
| Protocolo base | TCP | UDP |
| Modelo usado para la comunicación | Publicación-Suscripción | Pregunta-Respuesta<br>Publicación-Suscripción |
| Nodo de comunicación | M:N | 1:1 |
| Consumo de energía | Mayor que CoAP | Menor que MQTT |
| RESTful | No | Sí |
| Número de tipos de mensaje usados | 16 | 4 |
| Tamaño cabecera | 2 Bytes | 4 Bytes |
| Tipos de mensaje | Asíncrono | Asíncrono & Síncrono |
| Fiabilidad | 3 Niveles de calidad de servicio<br>QoS 0: Entrega no garantizada<br>QoS 1: Confirmación de entrega<br>QoS 2: Doble confirmación de entrega | Mensajes confirmables<br>Mensajes no confirmables<br>ACKs<br>Retransmisiones |
| Implementación | Fácil de implementar<br>Complejo para añadir extensiones | Pocas librerías existentes y soporte |
| Seguridad | No definida<br>Puede utilizar TLS/SSL | DTLS o IPSec |
| Otros | Útil para conexiones en localizaciones remotas<br>Sin gestión delerror | Baja saturación<br>Baja latencia<br>Problemas en NAT |

*Tabla 2:* [Comparación MQTT vs CoAP]. Fuente: https://www.pickdata.net/es/noticias/mqtt-vs-coap-mejor-protocolo-iot

Debido a que el software seleccionado posee soporte para el protocolo MQTT y a día de redacción de esta memoria se pueden encontrar mayor cantidad de proyectos y información de desarrollos IoT mediante el protocolo MQTT, se ha decidido tomar el protocolo MQTT como el protocolo de transmisión de datos para el nodo físico hacia la Nube.

Para la implementación de MQTT se utilizará el bróker **Eclipse Mosquitto**, que es la solución de código abierto más utilizada en desarrollos IoT.

### 3.2.3 La Nube. Gestión de datos

La decisión a tomar en la gestión de datos consiste en decidir si utilizar una plataforma IoT que brinde de ciertas funcionalidades propias de la plataforma o utilizar una herramienta IoT que permita la gestión de datos y sea compatible con los protocolos necesarios para hacer funcionar el sistema.
Para realizar un desarrollo más completo y personalizable, se ha decidido utilizar **Node-RED**, que nos brinda de una serie de ventajas:

- Es un entorno de programación visual, mediante nodos, por lo que cada nodo contiene un bloque de código a ejecutar, de manera que con conocimientos básicos de JavaScript y programación orientada a objetos, se puede llegar a realizar desarrollos que entran en otros ámbitos de la programación.
- Posee nodos que permiten ejecutar la mayoría de protocolos con el único requerimiento de que se especifique su funcionalidad.
- El número de nodos utilizables es ampliable, ofreciendo hasta más de 60.000 posibilidades distintas de manera oficial.
- Es muy cómodo de utilizar y ofrece gran variedad de posibilidades para resolver un mismo problema.
- Dado el uso de Docker para el despligue del servidor que contendrá el bróker MQTT, se podrá incluir también el servidor de Node-RED, ambos ejecutándose de manera local.

### 3.2.4 La Nube. Base de datos

Referente a la base de datos, se ha utilizado **Firebase Real-Time Database**:

- Al querer desarrollar una aplicación móvil como interfaz de usuario, esta base de datos está pensada específicamente para realizar este tipo de proyectos.
- Es un servicio en la nube que admite el uso de su propia web API para la transmisión de datos mediante el protocolo HTTP (desde Node-RED).
- No se requiere de disponibilidad crítica de la base de datos, ya que la interfaz de usuario está destinada a consulta y monitorización.
- Para este proyecto es suficiente con una sola instancia en la base de datos.
- Firebase y Flutter son desarrollados por la misma compañía (Google), por lo que constantemente están actualizando y renovando tanto el SDK de Flutter como los paquetes y dependencias de desarrollo de Firebase.
- Posee reglas de seguridad programables y personalizables.
- Desde la consola del proyecto donde se encuentra la base de datos, se puede gestionar el acceso para requerir autentificación por correo, permitir usuarios anónimos, contraseñas o realizar registros con cuentas de otras plataformas y redes sociales.

## 3.3 Interfaz de usuario

Para el desarrollo de la interfaz de usuario, finalmente se ha decidido desarrollarla en **Flutter**, como se ha mencionado en el apartado anterior. Los motivos han sido:

- En caso de en un futuro querer utilizar la misma aplicación en otros sistemas operativos como iOS o ejecutarla a través de un navegador Web, utilizar un lenguaje multiplataforma es lo ideal, ya que en lugar de desarrollar la aplicación desde cero, bastará con configurar el proyecto o crear un nuevo proyecto con el mismo código, pero con las especificaciones para ser ejecutado en otro sistema operativo.
- Debido a que el entorno de Firebase y Flutter están desarrollados por Google, existen paquetes oficiales y widgets de Flutter que amplían y facilitan mucho el desarrollo de la aplicación.
- Realizar una aplicación en Flutter otorga una libertad de decisión y flexibilidad de desarrollo al usuario, que si se hubiera utilizado la interfaz de usuario o dashboard de una plataforma IoT, se hubiera visto limitada.

## 3.4 Esquema de tecnologías seleccionadas

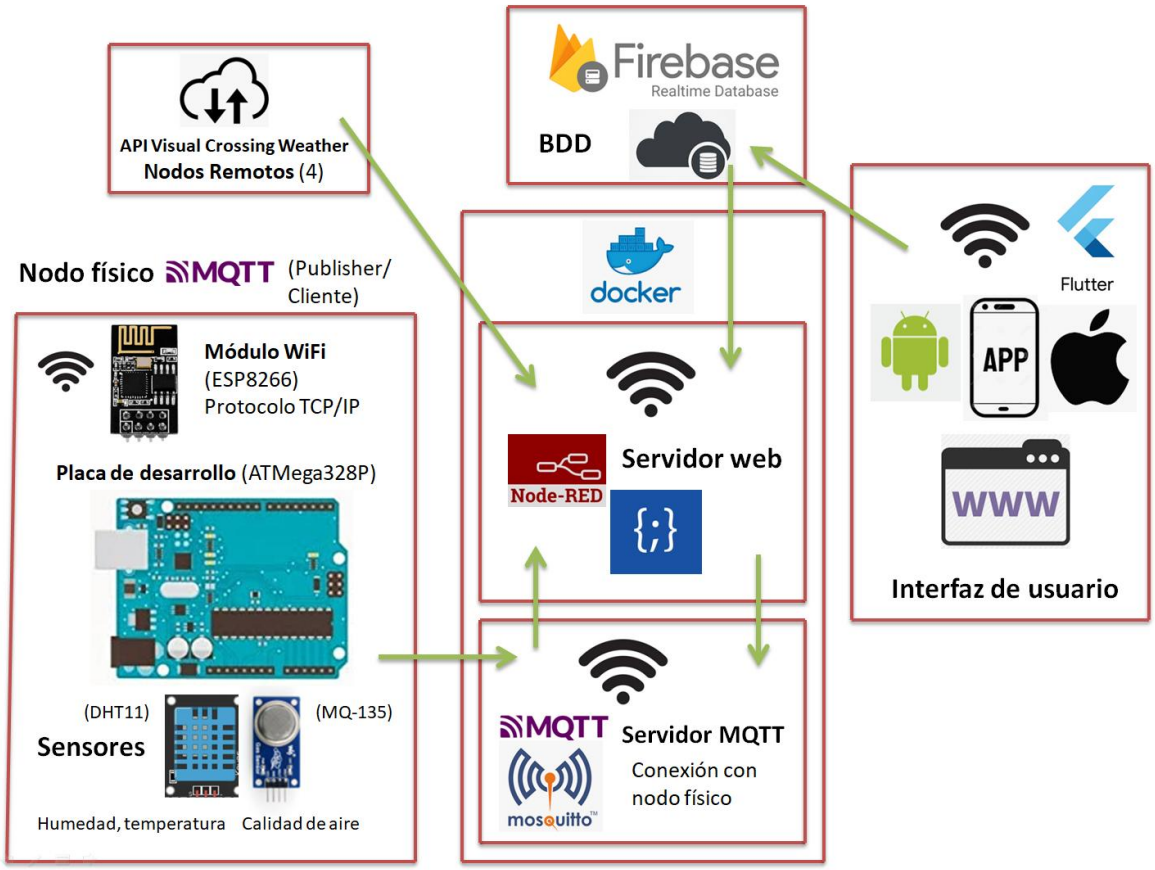


*Figura 2:* Rafael C. (2022). [Esquema desarrollo tecnologías seleccionadas]

# 4 Metodología

La metodología empleada en el desarrollo del proyecto y la obtención de una solución tecnológica ha consistido en tomar como referencia los elementos que componen una plataforma IoT y separarlos en paquetes de trabajo a desarrollar.

## 4.1 Nodos

### 4.1.1 Nodo físico

• **Pasos a seguir**

1- Realizar montaje del circuito.
2- Desarrollar el programa a ejecutar en el microcontrolador.
3- Comprobar el correcto funcionamiento del circuito.
4- Validar el correcto funcionamiento del sistema.

• **Técnicas utilizadas**

- Autentificación de usuario.
- Uso de protocolo de transmisión de datos.
- Envío de datos a través de protocolo de red WiFi.
- Formato de envío de datos tipo JSON String sin llaves.
- Librerías para protocolo de red, protocolo de transmisión, sensores y registros de microcontrolador.

• **Herramientas necesarias**

- Entorno de desarrollo. Microchip Studio y Arduino IDE.
- Placa de desarrollo. Arduino UNO/Genuino UNO
- Sensores. DHT11 y MQ-135.
- Módulo WiFi. ESP8266 ESP-01.

• **Esquema de funcionamiento**

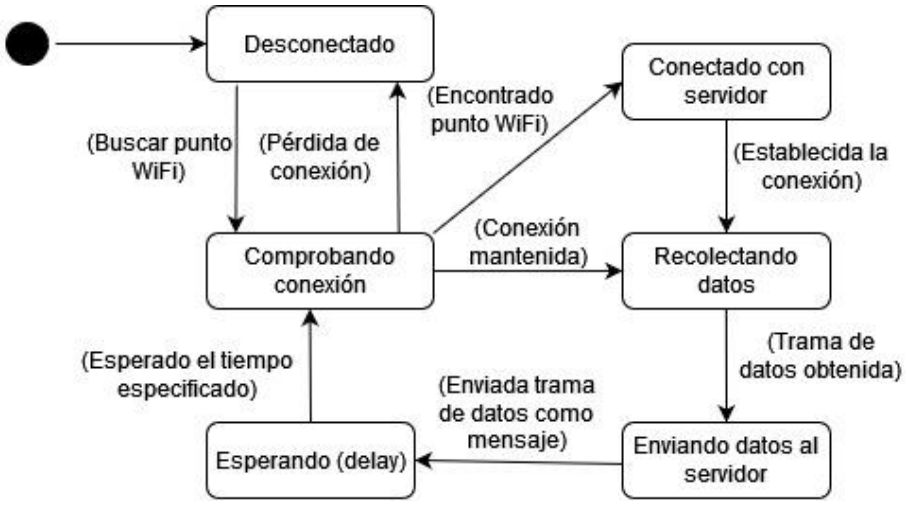


*Figura 3:* Rafael C. (2022). [Diagrama de estados: Software del Nodo]

### 4.1.2 Nodo remoto

• **Pasos a seguir**

1- Pedir una clave de acceso a la API de la base de datos.
2- Establecer protocolo para realizar peticiones a la API.
3- Filtrar y dar formato a la información obtenida de la base de datos pública.
4- Generar objeto JSON que contenga los datos.
5- Almacenamiento del objeto en la base de datos alojada en la nube.
6- Verificar el correcto almacenamiento de los datos en la base de datos.

• **Técnicas utilizadas**

- Acceso a base de datos pública.
- Uso de protocolo de transmisión de datos.
- Envío y recepción de datos gestionados en la nube.
- Funciones intermedias para filtrar y dar formato a los datos.

• **Herramientas necesarias**

- Base de datos pública. Visual Crossing Weather.
- Servidor Node-RED.
- Base de datos. Firebase Real-Time Database.

• **Esquema de funcionamiento**

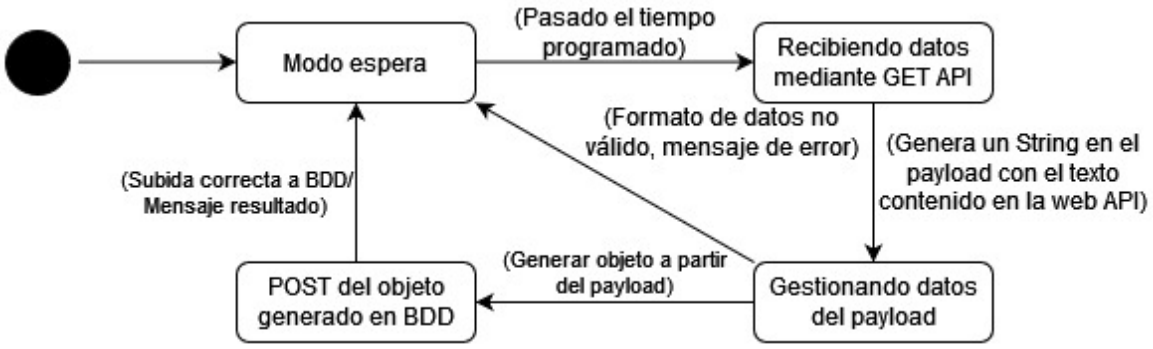


*Figura 4:* Rafael C. (2022). [Diagrama de estados: Nodo remoto]

## 4.2 Red de comunicaciones y la Nube

### 4.2.1 Comunicación. Protocolo de transmisión

- **Pasos a seguir**

1- Programar software para desarrollar el protocolo de transmisión de datos.
2- Instalar software que permita el despliegue y gestión de servidores.
3- Configurar el software para admitir usuarios anónimos o pedir autentificación.
4- Desarrollo del servidor para desplegar el protocolo de transmisión de datos.
5- Agregar especificaciones de red al servidor.
6- Desplegar (ejecutar y compilar) el software que forma el servidor contenedor del protocolo.
7- Validar correcto funcionamiento del servidor.

- **Técnicas utilizadas**

  - Autentificación de usuario.
  - Uso de Docker Desktop (aplicación de escritorio) para un despliegue más rápido y cómodo del servidor.
  - Uso de DockerFile y Docker-Compose para el despliegue del servidor mediante entornos de programación.
  - Imagen Mosquitto.

- **Herramientas necesarias**

  - Entorno de desarrollo. Visual Studio Code.
  - Docker SDK.
  - Docker Desktop.
  - Eclipse Mosquitto MQTT bróker.

- **Esquema de funcionamiento**

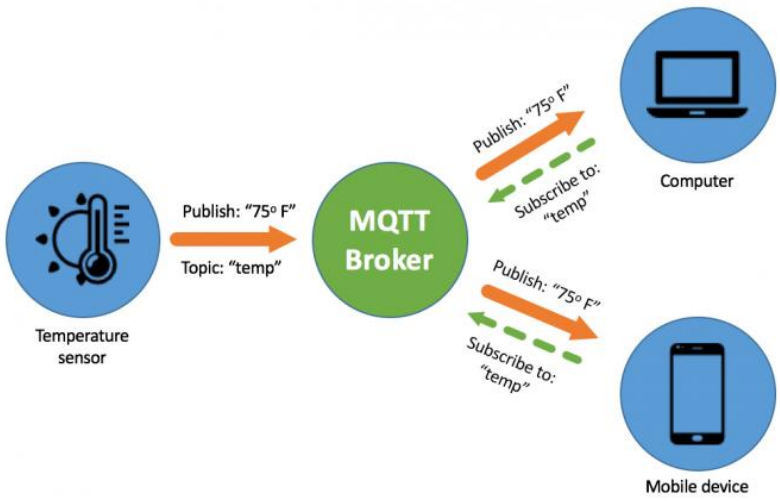


*Figura 5:* Vicente Lahoz (2019). [Ejemplo de comunicación mediante MQTT]. Security ArtWork. URL: https://www.securityartwork.es/2019/02/01/el-protocolo-mqtt-impacto-en-espana/

### 4.2.2 La Nube. Gestión de datos

- **Pasos a seguir**

1- Desarrollar software que permita el despliegue y gestión de servidores.
2- Despliegue del servidor que contiene la interfaz de usuario en web para la gestión de datos.
3- Validar correcto funcionamiento del servidor.
4- Recepción de datos del nodo físico mediante protocolo MQTT.
5- Añadir fecha y dar formato a los datos obtenidos.
6- Generar objeto JSON conteniendo los datos.
7- Almacenar el objeto en la base de datos alojada en la nube mediante protocolo HTTP.
8- Implementar nodos remotos.
9- Comprobar las conexiones entre servidores y base de datos.

- **Técnicas utilizadas**

  - Nodo MQTT in.
  - Protocolo HTTP (HTTP GET y POST en nodos http request).
  - Uso de nodos de función.
  - Uso de Docker Desktop (aplicación de escritorio) para un despliegue más rápido y cómodo del servidor.
  - Uso de DockerFile y Docker-Compose para el despliegue del servidor mediante entornos de programación.
  - Imagen Node-RED.

- **Herramientas necesarias**

  - Entorno de desarrollo. Visual Studio Code (+ Extensión Docker).
  - Docker SDK.
  - (Opcional) Docker Desktop.
  - Servidor (computadora destinada a este fin).
  - Navegador web.
  - Node.js.
  - NPM (Node Package Manager)

### 4.2.3 La Nube. Base de datos

• **Pasos a seguir**

1- Crear proyecto en la consola de Firebase y inicializar la base de datos Real-Time Database.
2- Establecer modelo de datos con el que se subirán los datos del nodo físico y nodos remotos.
3- Crear las rutas donde se alojarán las direcciones web de almacenamiento (distintas API) para cada tipo de nodo.
4- Realizar prueba e inicialización de cada una de las rutas a través del protocolo HTTP.
5- Añadir fecha y dar formato a los datos obtenidos.
6- Generar objeto JSON conteniendo los datos.
7- Almacenar el objeto en la base de datos alojada en la nube.

• **Técnicas utilizadas**

- Protocolo HTTP.
- Web API (URL).
- Base de datos NoSQL.

• **Herramientas necesarias**

- Firebase Real-Time Database.

## 4.3 Interfaz de usuario

- **Pasos a seguir**

1- Maquetación y diseño de las primeras pantallas.
2- Validar funcionamiento de las pantallas inicializando variables locales.
3- Vincular la aplicación con la base de datos.
4- Comprobar que la conexión con la base de datos se ha realizado correctamente utilizando las pantallas con datos extraídos de la base de datos.
5- Implementar el resto de pantallas.
6- Validar correcto funcionamiento.

- **Técnicas utilizadas**

  - Base de datos. Instancia con Firebase Real-Time Database.
  - Uso de paquetes de autentificación. Firebase Auth.
  - Sistema de navegación entre pantallas. Rutas nombradas.
  - Ficheros de datos y funciones.
  - Uso de paquetes con widgets extra.
  - Desarrollo de widgets propios.
  - Uso de variables locales y globales.

- **Herramientas necesarias**

  - Entorno de desarrollo. Visual Studio Code (Extensión Flutter y Dart).
  - Flutter SDK.
  - Dart SDK.
  - Dispositivo Android.

# 5 Desarrollo

En este capítulo se ha realizado una breve explicación de cómo se ha desarrollado el sistema, siguiendo la enumeración utilizada en "*Metodología"*.

Las validaciones y pruebas de funcionamiento han sido realizadas en el capítulo "*Pruebas de validación"*.

## 5.1 Desarrollo de los nodos

El desarrollo de los nodos se ha separado en tres partes:

1- El montaje de la circuitería del nodo físico.
2- Desarrollo del software del nodo físico.
3- Desarrollo de los nodos remotos.

### 5.1.1 Montaje del nodo físico

En primera instancia, se ha desarrollado el esquema de conexión para cada uno de los componentes que conforman el nodo físico.

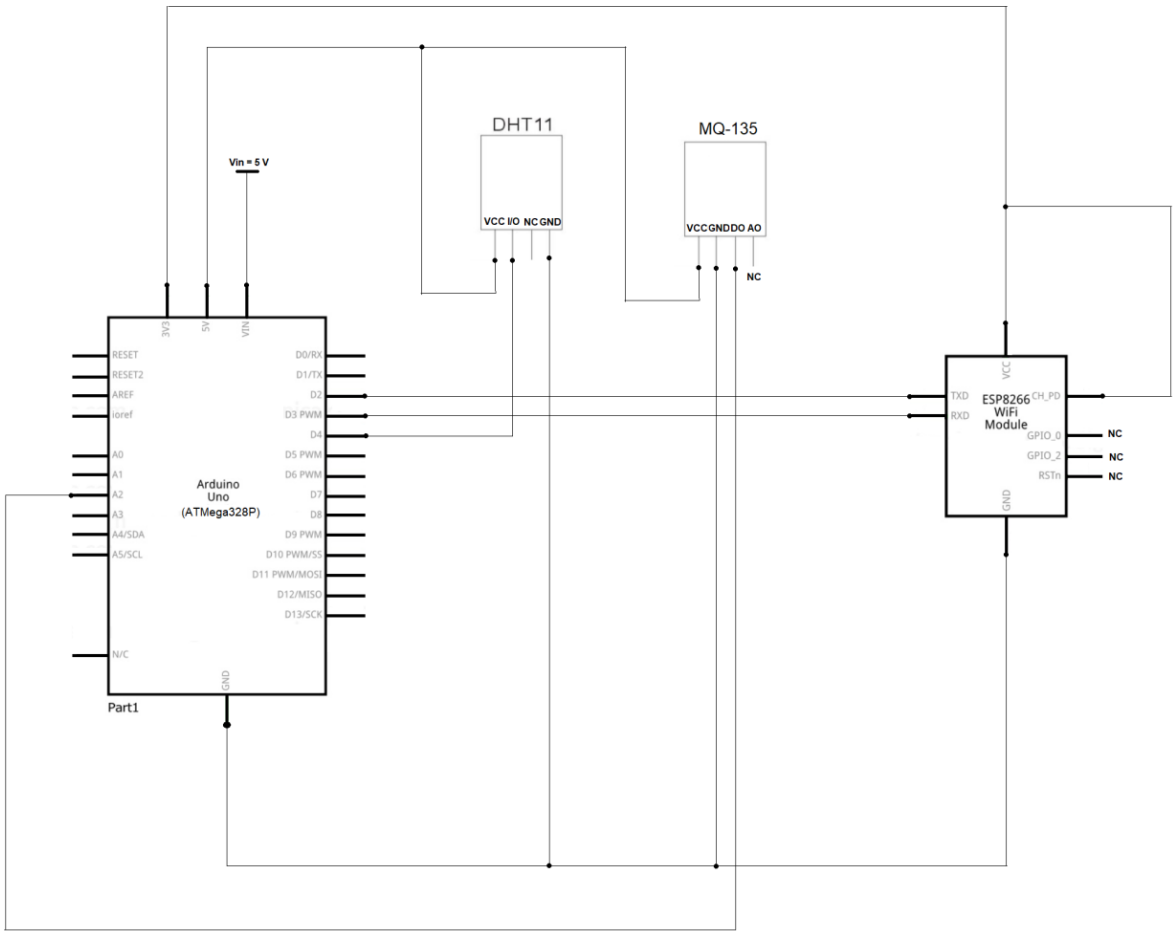


*Figura 6:* Rafael C. (2022). [Esquema eléctrico del nodo físico]

Una vez realizado el montaje del circuito siguiendo el esquema eléctrico en placa de baquelita y realizada la soldadura, el nodo físico tiene el siguiente aspecto:

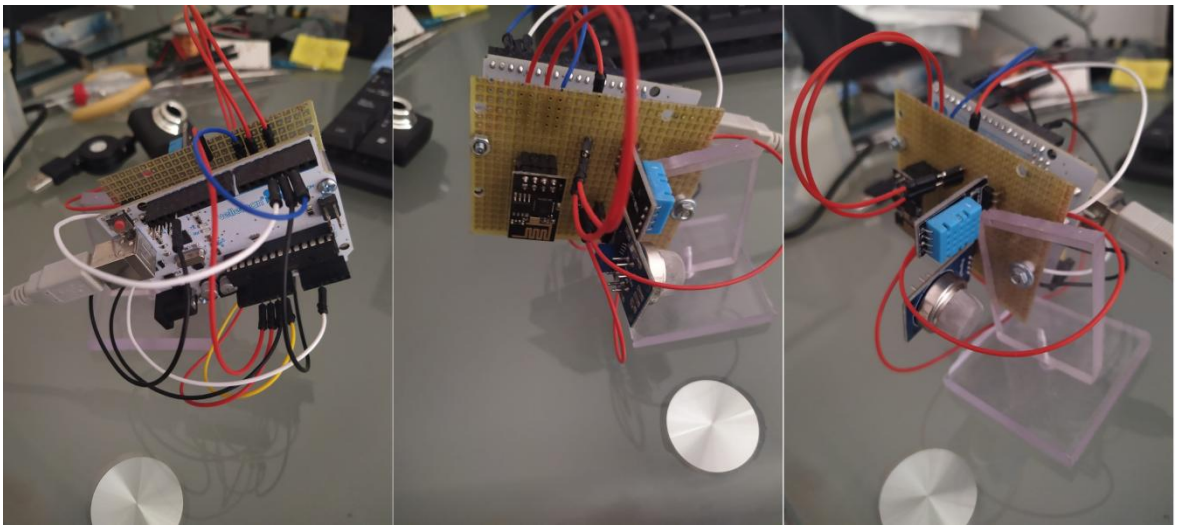

*Figura 7*: Rafael C. (2022). [Fotografía: Aspecto del nodo físico]

### 5.1.2 Desarrollo del software del nodo físico

Una vez montada la circuitería que conforma el nodo físico, habrá que desarrollar y descargar el código que ejecutará el microcontrolador de la placa de desarrollo, el código está basado en el diagrama de estados *Figura 2*. La secuencia de ejecución del código seguirá el siguiente algoritmo:

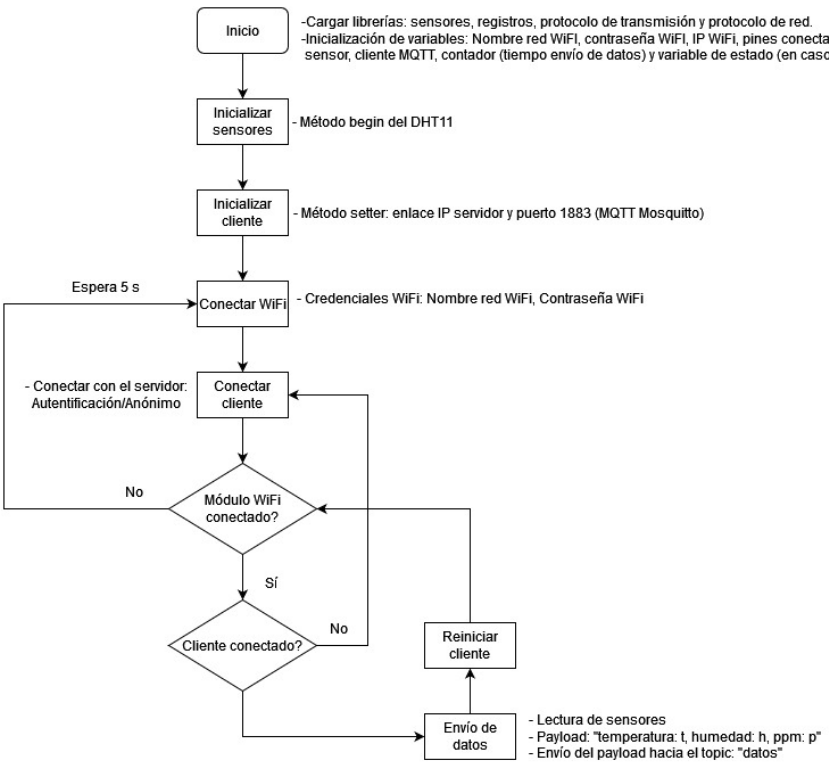


*Figura 8:* Rafael C. (2022). [Algoritmo código nodo físico]

Como se puede ver en el algoritmo, el diseño del software consiste en un bucle infinito en el que, por cada iteración, el cliente envía las mediciones realizadas con los sensores hacia un servidor mediante el topic (tema) datos.

Visto desde el punto de vista del bróker MQTT, lo que está realizando el nodo físico consiste en conectar como cliente al bróker través del puerto 1883 (conexión mediante protocolo TCP/IP) mediante la red WiFi. Una vez establecida la conexión con el servidor, el bróker queda en "escucha" del cliente, el cual realiza un envío al topic "datos" acompañado del payload con las mediciones tomadas de los sensores.

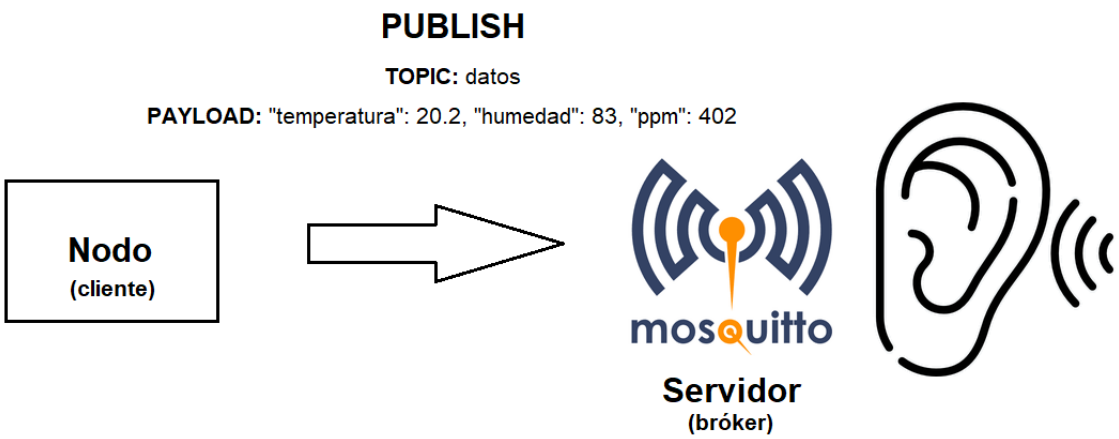


*Figura 9:* Rafael C. (2022). [Esquema software nodo físico - servidor MQTT]

En este desarrollo, se ha optado por juntar todas las mediciones en una sola cadena de caracteres siguiendo la notación JSON sin corchetes, de manera que todas las mediciones se enviarán en un mismo payload.

```
b'Enviando Datos\r\n'
b'"humedad":94.00,"temperatura":21.10,"ppm":395.00\r\n'
```

*Figura 10:* Rafael C. (2022). [Ejemplo payload enviado hacia servidor MQTT]

Hay muchas formas de desarrollar la parte de software en cuanto al formato de envío de los datos, desde la que se ha realizado en este proyecto, enviar distintos topics (uno por cada medición, por ejemplo), en el formato únicamente enviar el número de la variable obtenida de la medición, etc.

### 5.1.3 Desarrollo de los nodos remotos

Los nodos remotos mediante Node-Red son sencillos de obtener, ya que, como se ha comentado en el apartado *4.1.2 Nodo remotos*, la notación que contienen los objetos de la API son en JSON y no contienen caracteres especiales a los que debamos prestar realmente atención.

Para obtener la API Key de Visual Crossing se necesita registro en su página web y seguir los pasos indicados en la guía de usuarios nuevos registrados. Simplemente se deberá incluir la clave asociada a la cuenta en la URL obtenida al solicitar el acceso a la API.

El contenido de la API que interesa obtener para el desarrollo de los nodos se encuentra en el atributo "*currentConditions*", del cual se extraerá la fecha ("*datetimeEpoch"*, representada en segundos), la medición de temperatura (*"temp"*, representada en grados Celsius) y la humedad (*"humidity"*, representado en tanto por ciento).

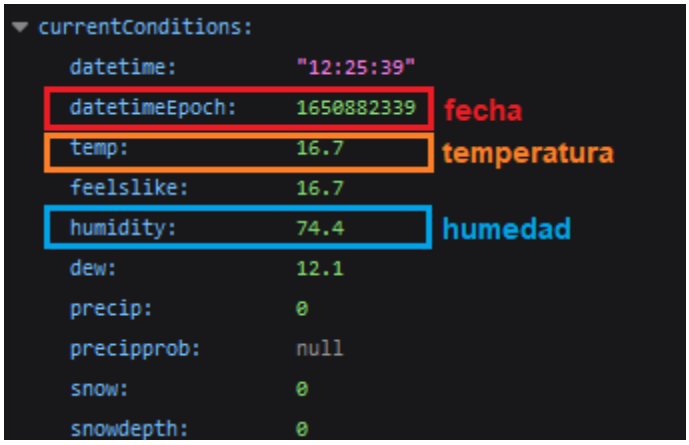


*Figura 11:* Rafael C. (2022). [Atributos a filtrar API Visual Crossing Weather]. Fuente: https://www.visualcrossing.com/

Siguiendo como referencia el esquema de funcionamiento del apartado *3.1.2 Nodos remotos*, la secuencia de desarrollo para el flujo en Node-RED es la siguiente:

- Nodo *timestamp*: Temporizador de repetición para generar un bucle que dispare la ejecución cada 20-30 minutos (tiempo aproximado en el que actualizan la API con las últimas mediciones).
- Nodo *http request*: Realizar una petición a la web API de la base de datos pública mediante el método HTTP GET, el cual nos devolverá el objeto en formato String.
- Nodo *json*: Dado que la cadena de caracteres obtenida ya está en formato JSON, con deserializarla se obtiene un objeto JSON que contiene toda la información obtenida de la web API.
- Nodos *function*: Filtrar y separar en variables los atributos seleccionados del objeto para generar la notación de nuevo objeto en formato String conteniendo únicamente los atributos seleccionados.
- Nodo *json*: Convertir el objeto en formato String a JSON Object.
- Nodo *join*: Node-RED permite generar un objeto a partir de dos o más objetos heredando los atributos de los objetos utilizados. De esta manera, se obtendrá el JSON Object listo para ser almacenado en la base de datos.
- Nodo *http request*: Se realiza una petición HTTP POST a la dirección donde se desea almacenar el objeto mediante la URL de la API correspondiente.

Siguiendo los pasos del desarrollo, el diagrama resultante para uno de los nodos remotos es el siguiente:

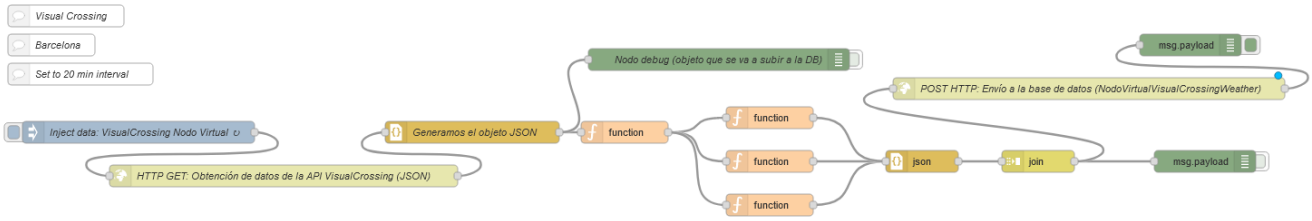


*Figura 12:* Rafael C. (2022). [Flujo del nodo remoto Barcelona (Node-RED)]

Cabe mencionar, que en este caso, la idea es obtener un objeto con pocas capas. Dado que la finalidad de estos objetos generados será guardarlos en una base de datos NoSQL en la nube, estas suelen tener problemas a la hora de realizar lecturas en modelos de datos con muchas capas.

## 5.2 Red de comunicaciones y la Nube

En este apartado se tratarán los siguientes temas:

- Comunicación. Desarrollo del servidor MQTT.
- La Nube. Desarrollo del servidor Node-RED.
- La Nube. Modelo de la base de datos.

Antes de tratar estos temas, es necesario recordar que, como se mencionó en el apartado *4.2.3 La Nube. Gestión de datos*, para el despliegue de los servidores MQTT (Eclipse Mosquitto bróker) y el servidor Node-RED se ha utilizado Docker a través del entorno de programación Visual Studio Code.

La secuencia utilizada para el despliegue de los servidores es la siguiente:

- Servicio MQTT: Añadir la última versión de Eclipse Mosquitto como servicio (a través de la imagen eclipse-mosquitto), exponiendo los puertos 1883 y 9001 (TCP/IP), dejándolos a disposición del servicio MQTT.
- Servicio Node-RED: Ejecutar un archivo para obtener la interfaz gráfica de programación de Node-RED (a través de la imagen node-red-dashboard), exponer el puerto 1880 al servicio Node-RED (proporcionará al usuario el acceso a la interfaz de Node-RED) y añadir una dependencia respecto al servicio MQTT para poseer acceso a ambos servicios de forma individual.

Al arrancar la secuencia, el equipo dejará expuesto el puerto 1880 para el servicio Node-RED y para el servicio MQTT utilizará el puerto 1883.

```
[info] Server now running at http://127.0.0.1:1880/
```

*Figura 13:* Rafael C. (2022). [Confirmación de arranque del servidor Node-RED]

```
Opening ipv4 listen socket on port 1883.
Opening ipv6 listen socket on port 1883.
mosquitto version 2.0.14 running
```

*Figura 14:* Rafael C. (2022). [Confirmación de arranque del servidor MQTT]

Una vez realizado el despliegue, se han vinculado los servidores a la aplicación Docker Desktop (versión de escritorio de Docker), ya que desde la propia interfaz de la aplicación de escritorio, permite revisar los registros de los servidores, consultar la información y ejecutar comandos de arranque o reinicio de los servidores. De esta forma, para iniciar los servidores, bastará con arrancarlos mediante un botón desde la propia aplicación, sin necesidad de utilizar comandos en la terminal.

### 5.2.1 Comunicación. Desarrollo del servidor MQTT

El desarrollo del servidor MQTT, que contendrá el protocolo de transmisión, consta de tres partes:

1- Puesta en marcha del servidor.
2- Comunicación con los clientes.
3- Publicación y suscripción al bróker.

En cuanto a la puesta en marcha del servidor, ya se ha detallado en el apartado anterior mediante el despliegue del software a través de la herramienta Docker. Es necesario mencionar que, para versiones de Mosquitto superiores a la 2.0.0, si se mantienen las especificaciones por defecto, el servidor se conectará al puerto 1883 y pedirá autentificación de usuario.

Una vez puesto en marcha el servidor, este quedará en "escucha" ocupando el puerto 1883. El puerto 1883 es el puerto que se suele utilizar por defecto para el uso del protocolo MQTT, cuyos mensajes no están cifrados y admite conexión anónima (para ver el resto de servidores a los que puede escuchar: *The server*. Mosquitto Test. URL: https://test.mosquitto.org/ ). Por lo tanto, cuando haya un dispositivo o servicio que envía una petición de acceso hacia el puerto y cumple con los requisitos del servidor MQTT, se establece una conexión en base a la pila TCP/IP para la comunicación entre el bróker MQTT y el dispositivo o servicio.

Una vez los clientes han sido conectados al bróker MQTT, los clientes podrán publicar y/o suscribirse a topics. Para este desarrollo, tenemos dos clientes, que son: el nodo físico (publicador) y el servidor Node-RED (suscriptor).

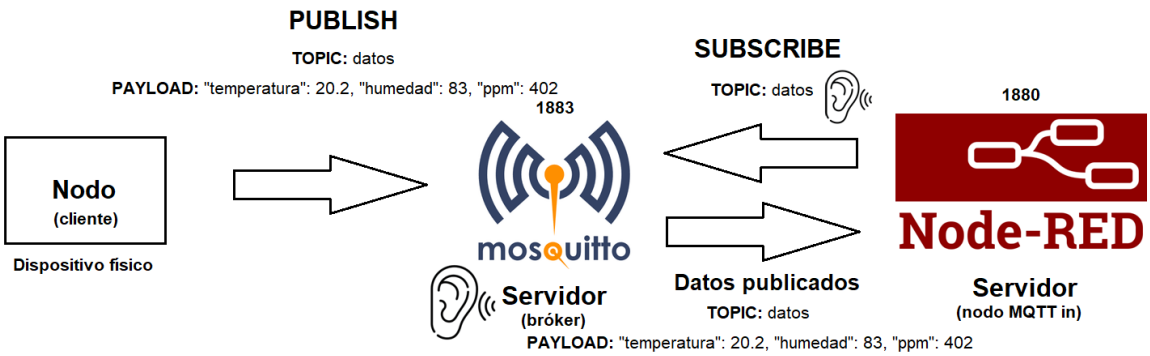


*Figura 15:* Rafael C. (2022). [Esquema de funcionamiento: Servidor MQTT]

### 5.2.2 La Nube. Desarrollo del servidor Node-RED

El servidor Node-RED, para este proyecto, es el encargado de gestionar y procesar los datos, estar suscrito al bróker MQTT, realizar peticiones a bases de datos mediante el protocolo HTTP y dar acceso a servicios en la nube.

En el entorno de Node-RED, el conjunto de nodos por los que se hacen pasar los datos, desde un inicio hasta su fin, se le suele llamar flujo ("*flow*"), aunque también suele referirse al archivo de exportación e importación del proyecto guardado.

Debido a que en este proyecto se desarrollan varios nodos, hay que realizar cinco flujos: cuatro para cada uno de los nodos remotos (flujo mostrado en *5.1.3 Desarrollo nodos remotos)* y uno para el nodo físico.

A continuación, se realizará un pequeño análisis del desarrollo del flujo utilizado en el nodo físico:

- Nodo *mqqt in*: Es el nodo con el que se realizará la suscripción al bróker MQTT, se le debe especificar la IP del servidor, puerto 1883, que realice una suscripción a un único topic y especificar el topic ("datos"). Una vez esté configurado, cuando el bróker reciba una publicación hacia ese topic, el nodo de suscripción a MQTT nos devolverá el payload de la publicación como un String.
- Nodo *function*: Gracias a la notación con la que se ha decidido enviar el payload, en este nodo bastaría con simplemente añadir corchetes o llaves de texto ["{", "}"] a inicio y final del payload para obtener un objeto serializable a JSON Object. Previo a este paso, se le añadirá la fecha mediante la clase Date como atributo extra del objeto a generar.
- Nodo *json*: Generamos el JSON Object con los atributos: temperatura, humedad, ppm y fecha.
- Nodo *http request*: Igual que con los nodos remotos tratados en el apartado *5.1.3 Desarrollo de los nodos remotos*: Se realiza una petición HTTP POST a la dirección donde se desea almacenar el objeto mediante la URL de la API correspondiente.


*Figura 16:* Rafael C. (2022). [Flujo del nodo físico]

### 5.2.3 La Nube. Modelo de la base de datos

Como se ha mencionado en los apartados anteriores, el almacenamiento de los datos obtenidos a partir de la red de sensores es realizado mediante peticiones POST del protocolo HTTP hacia la API de Real-Time Database. Al realizar este tipo de implementación, para este tipo de operaciones, no influimos en la capacidad de respuesta de la base de datos, por lo que a la hora de monitorizar nuevos datos, el tiempo transcurrido desde la lectura de mediciones por parte del nodo físico, será lo que demoren en ejecutar los protocolos de transmisión.

#### 5.2.3.1 Modelo de datos

En los apartados anteriores del desarrollo de flujo en Node-RED se ha comentado la notación y el formato en el que se irán almacenando los objetos que contienen los datos de las mediciones. Es importante tener claro, para el desarrollo de la base de datos y posterior uso en la aplicación, cómo se van a estructurar los datos:

**• Nodo físico:**

- Fecha. El tipo de dato es una cadena de caracteres en formato ISOString. Representa la fecha y hora en la que el servidor de Node-Red ha recibido el mensaje del nodo físico a través del servidor MQTT tras haber realizado la medición.
- Temperatura. Dato de tipo real, admitiendo decimales. Representa el valor numérico de la medición de temperatura tomada mediante el sensor DHT11 del nodo físico.
- Humedad. Dato de tipo entero. Representa el valor numérico de la medición de humedad tomada mediante el sensor DHT11 del nodo físico. El dato se guarda en la base de datos como entero, pero posteriormente se utilizará como real en la aplicación. Esto es debido a que las mediciones de los nodos remotos poseen mayor precisión en la medición, ofreciendo un grado de magnitud mayor.
- PPM: Dato de tipo entero. Representa el valor numérico de la medición de temperatura tomada mediante el sensor MQ-135 del nodo físico.

**• Nodos remotos:**

- Fecha. El tipo de dato es una cadena de caracteres en formato ISOString. Representa la fecha y hora de la última medición que nos ofrece la web API de la cual obtenemos los datos.
- Temperatura. Dato de tipo real, admitiendo decimales. Representa el valor numérico de la medición de temperatura tomada de la localización geográfica obtenida de la web API.
- Humedad. Dato de tipo real, admitiendo decimales. Representa el valor numérico de la medición de humedad tomada de la localización geográfica obtenida de la web API.

#### 5.2.3.2 *Estructura de la base de datos*

Dado que la base de datos no es relacional (NoSQL), en lugar de seguir un modelo de datos relacional, hay que diferenciar por documentos semiestructurados (objetos formados mediante particiones de estructuras de datos). Por lo tanto, el formato en el cual almacenan los datos en la base de datos Real-Time Database de Firebase, sigue el siguiente esquema:

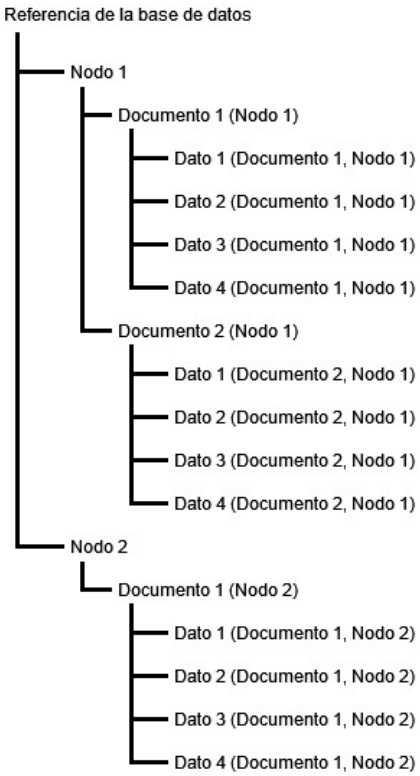


*Figura 17:* Rafael C. (2022). [Esquema de la estructura de la base de datos]

Para ejemplificar y aclarar los contenidos del esquema, se adjunta una captura de pantalla de la base de datos:


*Figura 18:* Rafael C. (2022). [Captura de la base de datos, medición del nodo físico]

## 5.3 Desarrollo de la interfaz de usuario

El contenido de este apartado, se separará en 4 subapartados:

- Lectura de la base de datos
- Funcionalidades de la aplicación
- Estructura de la aplicación
- Prototipos de pantallas

### 5.3.1 Lectura de la base de datos

En primera instancia, se debe registrar la aplicación con Firebase para uno de los sistemas operativos (Android, Web o iOS), añadiendo el fichero de servicios de Google en el fichero de la plataforma la cual vayamos a desarrollar la aplicación, en este caso, dentro de la carpeta de Android, junto al compilador.

A continuación, hará falta agregar las dependencias para utilizar los servicios de Google al compilador de Android y modificar la configuración por defecto del compilador para que la versión del SDK mínima sea compatible con los paquetes y servicios.

Finalmente, se añaden los paquetes necesarios para obtener acceso a las librerías específicas para Firebase Real-Time Database: firebase_database y firebase_auth (en caso de que se haya activado la autentificación en la consola de Firebase).

Una vez configurada la aplicación para trabajar con Firebase y vinculada a la base de datos, la lectura de la base de datos se realiza a través de una referencia hacia la base de datos. En este caso, además de obtener la referencia, se ha realizado un filtrado para obtener la información del último documento registrado en la ruta NodoFisico a través de una query (queda en escucha y se actualiza a tiempo real):

```
var _ref = FirebaseDatabase.instance.ref().child("NodoFisico").limitToLast(1);
```

*Figura 19:* Rafael C. (2022). [*Query* del último registro del nodo físico]

Entrando un poco más en detalle, para obtener el objeto JSON de la query, en este caso se ha utilizado el widget *FirebaseAnimatedList*, que devuelve un widget *ListTile* que contiene un *DataSnapshot* para cada uno de los documentos a los que apunta la query ordenados por índice (en este caso, únicamente hay un solo elemento, por lo que lo que obtenemos es el objeto JSON que enviamos en la última medición del nodo físico).

### 5.3.2 Funcionalidades de la aplicación

1. Problema

Se requiere obtener una interfaz de usuario interactiva en móviles Android para poder visualizar los datos obtenidos a partir de la información dada por distintos nodos, almacenada en una base de datos en Firebase.

2. Solución

Crear una aplicación que permita la visualización de los datos almacenados, realizar un filtrado en función de la localización de los nodos y en función de sus fechas.

3. Funcionamiento general

Acceso al servicio en la nube:

La aplicación estará conectada a un servicio en la nube (Firebase) para vincular a los usuarios de la aplicación con la base de datos. Al arrancar la aplicación, se establecerá dicha conexión al servicio y obtendremos una referencia de acceso a la base de datos a partir de la cual extraeremos la información.

Navegación entre pantallas:

El modelo de enlace y navegación entre pantallas será mediante rutas nombradas.

Acceso a la información de los nodos:

- Mediante el muestreo de una lista con el nombre de todos los nodos a los que se puede acceder pulsando sobre el nombre del nodo.
- A través de un mapa donde se mostrarán los puntos geográficos donde están situados los nodos recolectores de información. Al pulsar sobre los nodos, se accederá a la información del nodo.

Información de los nodos:

Cuando accedemos a la información de un nodo, nos dirigiremos a la pantalla del nodo, donde podremos ver la fecha y última medición que se ha realizado en dicho nodo. La información de la última medición se actualizará en tiempo real.

En esta misma pantalla, se podrá acceder a distintas pantallas que poseerán una lista de todas las mediciones registradas en la base de datos.

Desde esta última pantalla también podremos acceder a una pantalla donde visualizar gráficos que representan la evolución temporal de las mediciones, los cuales se podrán filtrar en función de la fecha seleccionada.

### 5.3.3 Estructura de la aplicación

Se deberá establecer una estructura simple y replicable para poder añadir nuevos nodos (físicos o remotos).

• Pantallas de navegación principal. Permiten seleccionar de qué nodo queremos consultar la información:

- Mapa de nodos
- Lista de nodos

• Pantallas de nodo:

En este proyecto, habrá dos tipos de pantallas de nodo, las pantallas de nodo físico y las pantallas de nodo remoto. En el proyecto original final, lo ideal es que únicamente se repliquen pantallas de nodo físico.

- Nodo físico
    - Datos de temperatura:
        - Lista de mediciones de temperatura
        - Gráfico de mediciones de temperatura vs tiempo
    - Datos de humedad:
        - Lista de mediciones de humedad
        - Gráfico de mediciones de humedad vs tiempo
    - Datos de calidad de aire:
        - Lista de mediciones de calidad de aire
        - Gráfico de mediciones de calidad de aire vs tiempo
    - Todas las mediciones:
        - Lista de mediciones de temperatura, humedad y calidad de aire
- Nodos remotos
    - Datos de temperatura:
        - Lista de mediciones de temperatura
        - Gráfico de mediciones de temperatura vs tiempo
    - Datos de humedad:
        - Lista de mediciones de humedad
        - Gráfico de mediciones de humedad vs tiempo
    - Todas las mediciones:
        - Lista de mediciones de temperatura, humedad y calidad de aire

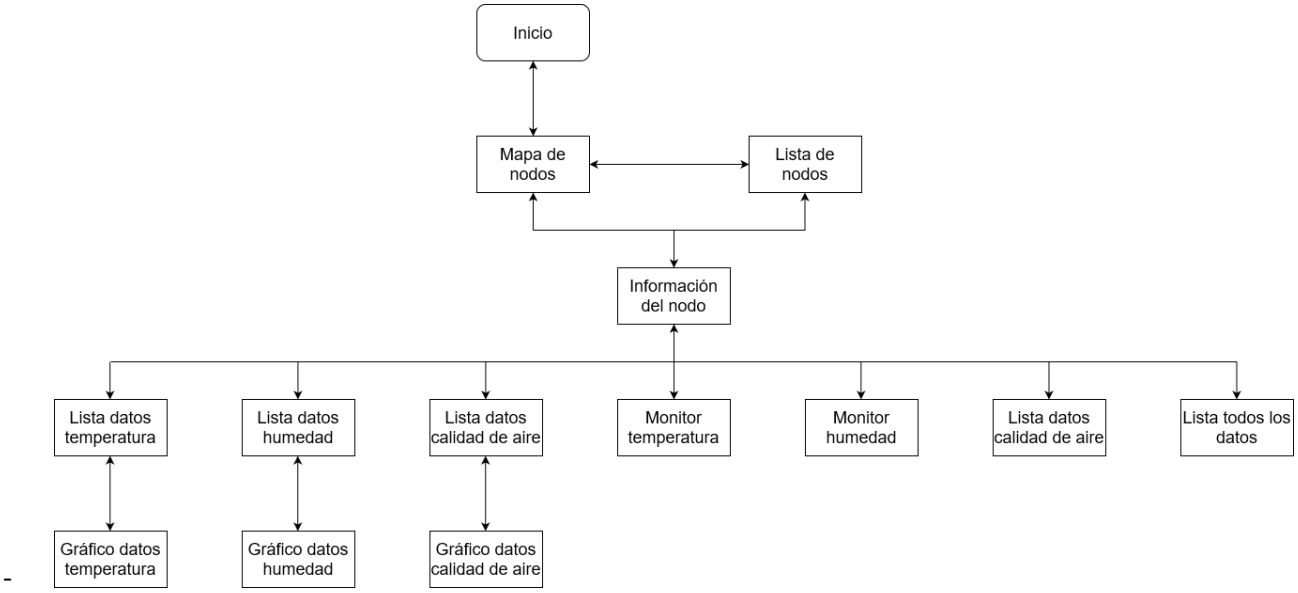


*Figura 20:* Rafael C. (2022). [Esquema de navegación entre pantallas]

### 5.3.4 Prototipos de pantallas

En este apartado se mostrarán algunos de los prototipos de las pantallas desarrolladas en Flutter para medicioness de temperatura, indicando la funcionalidad de cada uno de los objetos interactivos de cada pantalla.


*Figura 21:* Rafael C. (2022). [Prototipo de Pantalla: Mapa de nodos]


*Figura 22:* Rafael C. (2022). [Prototipo de Pantalla: Lista de Nodos]

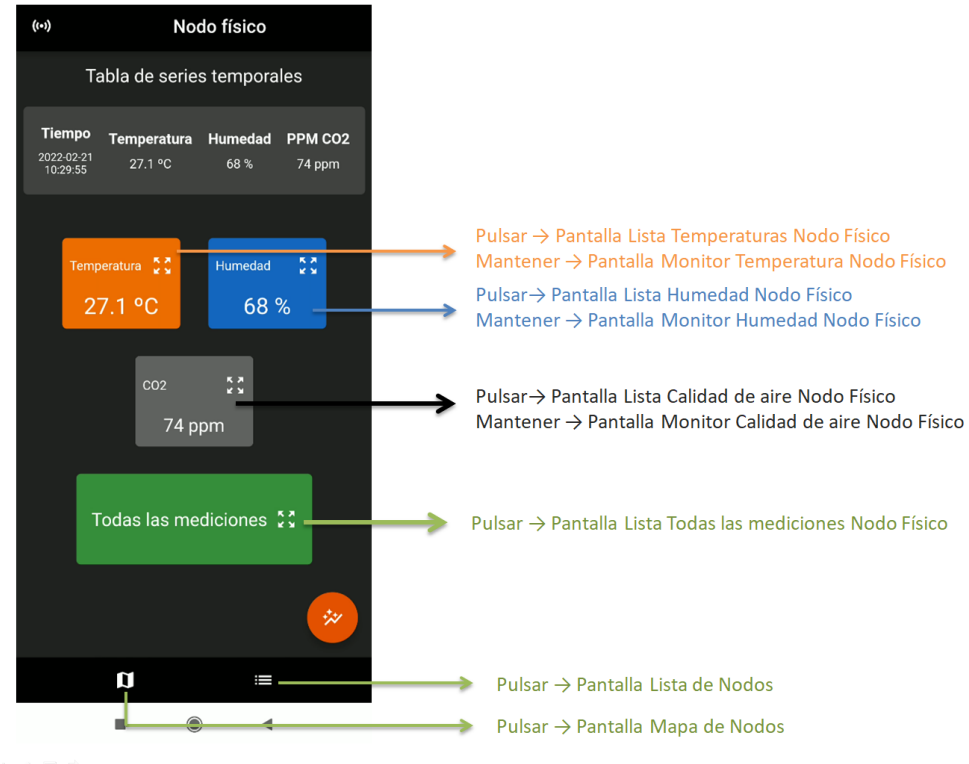


*Figura 23:* Rafael C. (2022). [Prototipo de Pantalla: Nodo]

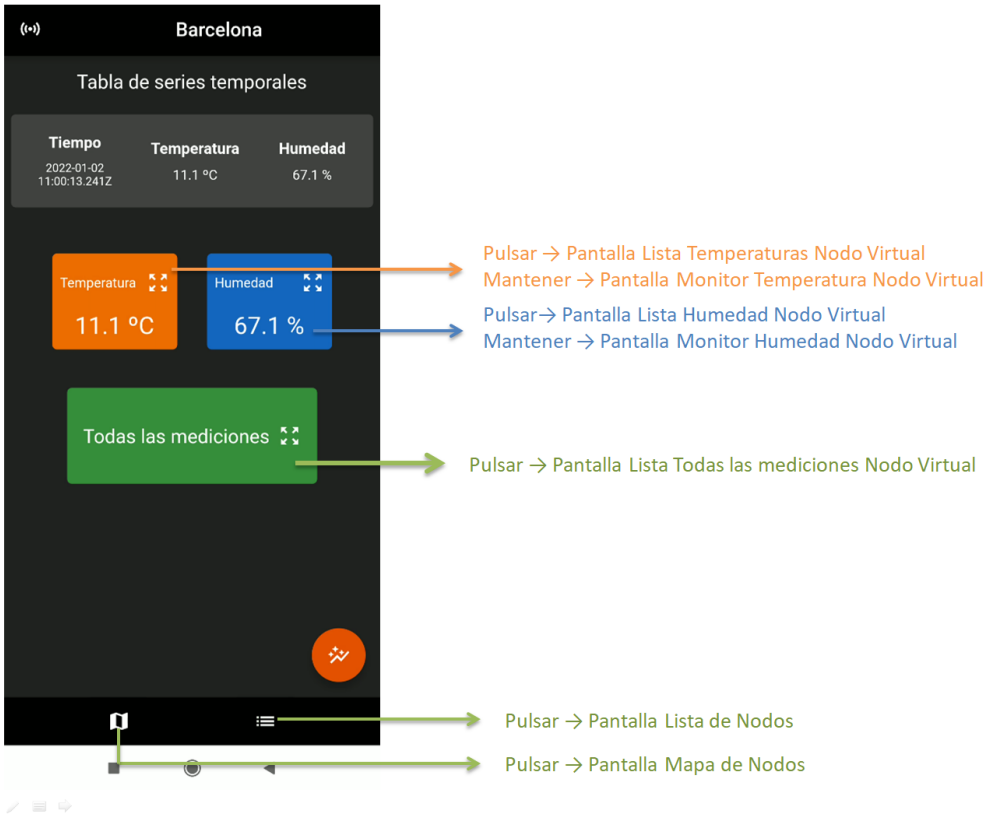


*Figura 24:* Rafael C. (2022). [Prototipo de Pantalla: Nodo remoto]


*Figura 25:* Rafael C. (2022). [Prototipo de Pantalla: Pantalla lista de temperatura]

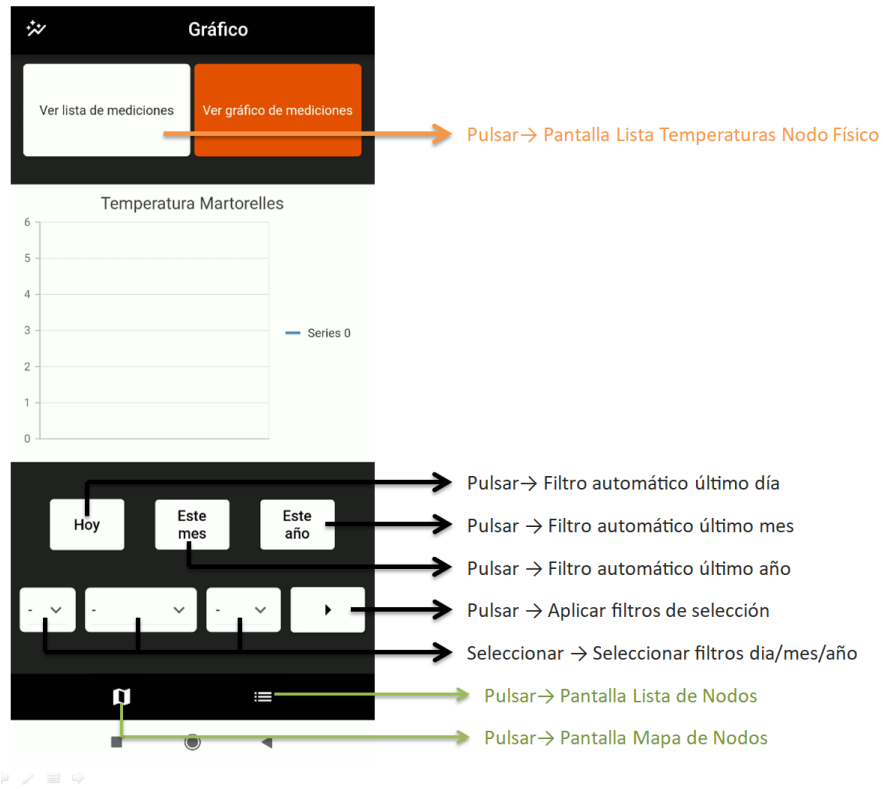


*Figura 26:* Rafael C. (2022). [Prototipo de Pantalla: Pantalla gráfico (temperatura)]

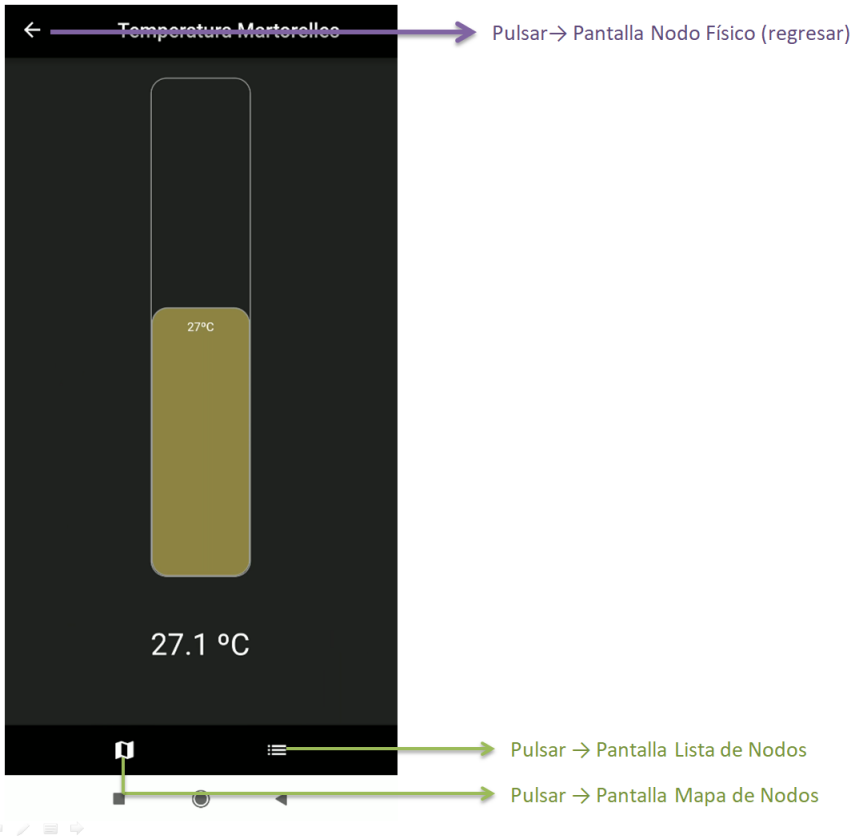


*Figura 27:* Rafael C. (2022). [Prototipo de Pantalla: Pantalla monitor temperatura]

# 6 Pruebas de validación

La comprobación del correcto funcionamiento del sistema, constará de un seguimiento de las etapas del sistema: desde la medición de los sensores hasta el muestreo de la medición en la interfaz de usuario.

## 6.1 Servidores

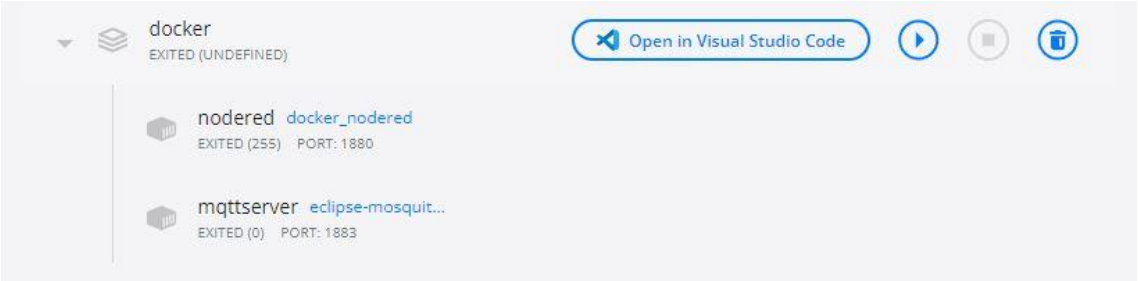


*Figura 28:* Rafael C. (2022). [Lista de servidores en Docker Desktop]

En primer lugar, para empezar a recibir datos, es necesario tener tanto el servidor MQTT que contiene el bróker con el protocolo de transmisión activo, junto con el servidor Node-Red, para realizar la recepción (publicación MQTT), gestión y envío de los datos hacia la base de datos en la nube.


*Figura 29:* Rafael C. (2022). [Lista de servidores en marcha]

Para asegurar que la conexión se ha efectuado correctamente, los servidores no están dando problemas identificación y/o alguno de los servidores está presentando problemas debido a nuevas actualizaciones, podemos revisar el terminal de ejecución de Docker (en caso de estar ejecutando los servidores directamente desde un IDE, revisar el terminal o consola del propio IDE).
Se revisarán los registros del terminal perteneciente al servidor MQTT, donde en la última línea se nos tiene que notificar que el servidor está funcionando. En caso de que haya algún error de conexión, el servidor tratará de reconectarse constantemente, dando como registro en el terminal el motivo por el que no se pudo conectar.

```
Opening ipv4 listen socket on port 1883.
Opening ipv6 listen socket on port 1883.
Error: Address not available
mosquitto version 2.0.14 running
```

*Figura 30:* Rafael C. (2022). [Registro del servidor MQTT]

A continuación, habrá que revisar los registros del terminal perteneciente al servidor donde está alojado Node-Red. En primera instancia, si aún no se ha configurado el servidor MQTT, interesará ver si el servidor está activo.

```
[info] Node-RED version: v2.1.4
[info] Node.js  version: v14.18.2
[info] Linux 5.10.102.1-microsoft-standard-WSL2 x64 LE
[info] Loading palette nodes
[info] Dashboard version 3.1.3 started at /ui
[info] Settings file  : /data/settings.js
[info] Context store  : 'default' [module=memory]
[info] User directory : /data
[warn] Projects disabled : editorTheme.projects.enabled=false
[info] Flows file     : /data/flows.json
[info] Server now running at http://[redacted]:1880/
```

*Figura 31:* Rafael C. (2022). [Registro del servidor Node-Red]

Si ya está configurado el servidor MQTT o ya tenemos un flujo creado en este servidor, deberá aparecer en las últimas líneas del registro tras el arranque de los servidores.

```
[info] Starting flows
[info] Started flows
[info] [mqtt-broker:localhost] Connected to broker: mqtt://[redacted]:1883
```

*Figura 32:* Rafael C. (2022). [Registro del servidor Node-Red: Conexión MQTT]

Para terminar de revisar y asegurar que la conexión con el bróker se ha realizado correctamente, simplemente será necesario abrir la interfaz de Node-Red en el navegador, colocar un nodo "mqtt in", configurar el nodo con las credenciales del servidor donde tenemos alojado el bróker y, en caso de que se haya realizado correctamente y no haya ningún problema de conexión, debajo del nodo debe confirmar la conexión.

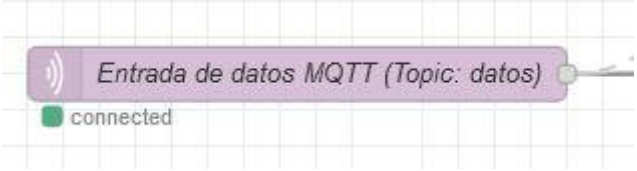


*Figura 33:* Rafael C. (2022). [Interfaz Node-Red: Nodo MQTT conectado al servidor]

## 6.2 Nodo físico

La validación del nodo físico, se realizará alimentando la placa mediante el puerto USB a través del computador. De esta forma la placa se podrá comunicar mediante conexión serial para visualizar qué está realizando el nodo en pantalla mediante las funciones de depuración añadidas en el programa del microcontrolador, destinadas para este mismo propósito.
Una vez conectada la placa al computador mediante el puerto USB, el microcontrolador empezará a ejecutar el programa. Para monitorizar el puerto serial, existen varias aplicaciones dedicadas a esta función como Serial Port Monitor (Windows) o CoolTerm (Multiplataforma) y entornos de programación o de diseño que incorporan esta función o la tienen como extensión como Microchip Studio, Arduino IDE o LabView. En este proyecto, se ha programado y utilizado un script en Python para obtener la lectura del puerto serial por consola cuando se ha requerido de esta función durante el desarrollo y pruebas.

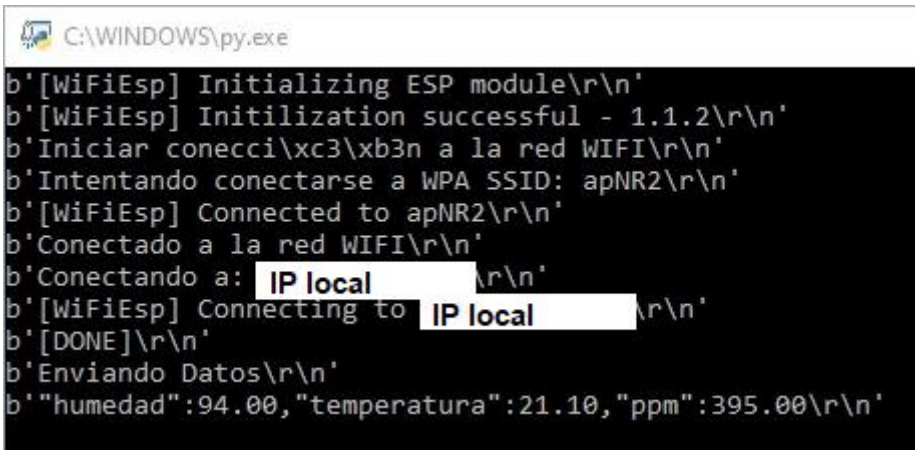


*Figura 34:* Rafael C. (2022). [Captura mensajería serial con placa de desarrollo]

Como se puede ver, el recorrido que realiza sigue el programa mencionado en el desarrollo del nodo físico. Primero realiza las funciones de configuración, donde se inicializan los periféricos (módulo WiFI y sensores), realiza la comprobación de que el módulo WiFi esté funcionando, conecta con la red WiFi de la IP local (en este caso) y envía los datos recolectados por los sensores al bróker MQTT. En caso de que haya algún tipo de error, enviará un mensaje del tipo "*[FAILED]: rc=-NUM*" donde NUM será el tipo de error (numeral) que habrá que analizar, en caso de que suceda.

## 6.3 Comunicación

La validación de la comunicación para este apartado ha sido realizada de dos formas, aunque existen varias maneras de realizar esta validación:

- Utilizando nodos debug en Node-Red, estos permiten visualizar en el terminal Debug si ha habido algún evento en el flujo y consultar el contenido del payload en el momento del evento. Para este caso, el nodo "*debug*" ha sido situado justo antes de ejecutar el nodo "*http request*" en el que se ejecutará el POST HTTP hacia la base de datos. Por lo que debe aparecer en el payload del mensaje es un objeto generado a partir de los datos obtenidos del sensor junto con la fecha (timestamp) en formato ISOStandard.


*Figura 35:* Rafael C. (2022). [Node-Red Debug: Payload a enviar a la BDD]

- Mediante la base de datos, al realizar el envío de datos desde el nodo, dado que el tiempo que tarda la información en recorrer el sistema y ser almacenada en la base de datos es inferior a un segundo, cabe la posibilidad de validarlo a partir del envío en la base de datos. El razonamiento de esta validación consiste en que, si no se ha podido realizar correctamente la comunicación, tampoco es posible que se haya generado un objeto a través de los servidores para ser enviado y almacenado. Por tanto, si el nodo ha realizado el envío de datos y se produce una escritura de un documento nuevo en la base de datos y el contenido del documento se corresponde al

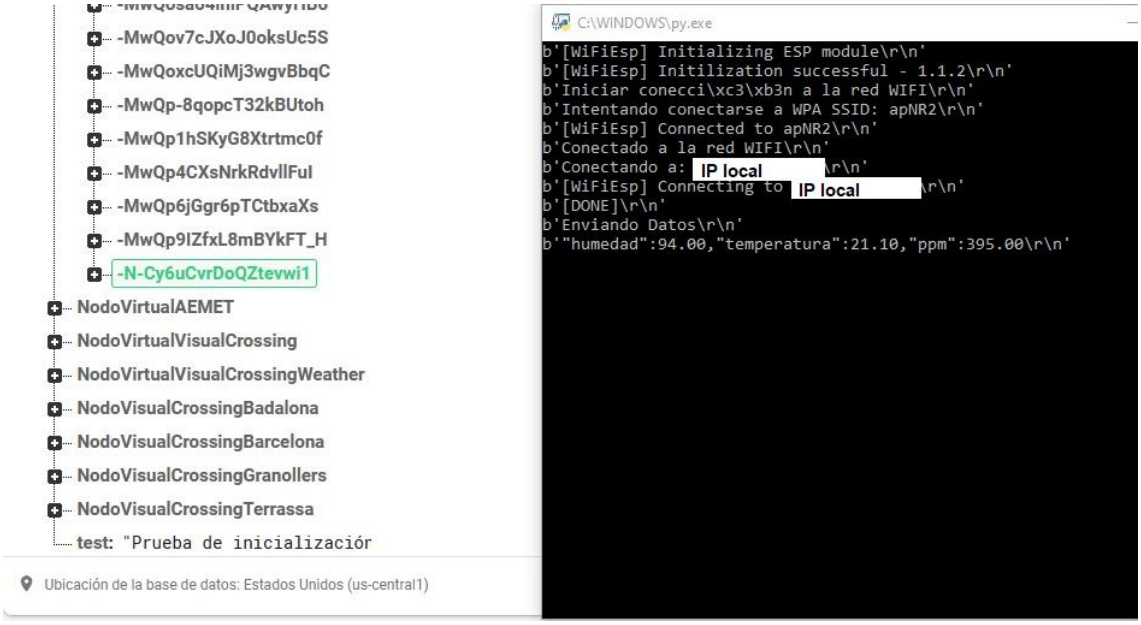


*Figura 36:* Rafael C. (2022). [Captura: escritura de nuevo documento en BDD]

de la medición realizada por el nodo, quedaría validada la comunicación entre dispositivos y servicios del sistema.


*Figura 37:* Rafael C. (2022). [Base de datos: Contenido del documento]

Otras opciones de validación consisten en revisar los registros de los servidores, para consultar en el terminal si el servidor MQTT ha realizado una nueva operación o, en el servidor Node-Red, si ha habido actividad nueva en el flujo.

## 6.4 Interfaz de usuario

Una vez validada la comunicación mediante la base de datos y conociendo que se almacenan correctamente los datos, los errores que pueden suceder en la interfaz de usuario son que la base de datos no contenga datos o bien que no establezca la conexión con la base de datos, ya sea por un fallo en el protocolo web del dispositivo de la interfaz de usuario o bien por algún fallo del servidor en la nube de Firebase.

En este caso, como se ha validado en los puntos anteriores, al consultar en la interfaz de usuario la pantalla donde está referenciada la instancia del nodo utilizado, podemos ver la medición actualizarse en tiempo real.


*Figura 38:* Rafael C. (2022). [Captura dispositivo móvil en Pantalla Nodo]


*Figura 39:* Rafael C. (2022). [Captura dispositivo móvil en Pantalla lista temperatura]

# 7 Resumen del presupuesto

Para calcular el presupuesto del desarrollo, hay que considerar los costos directos, costos indirectos.

Costos directos

En los costos directos se incluyen los costos de mano de obra y materiales:
• Los costos de mano de obra. Para calcular los costos de mano de obra, se ha establecido una relación entre las horas indicadas para la realización del proyecto

$$300\ h = 12\ créditos \cdot 25 \frac{horas}{crédito}$$

y el sueldo medio anual de un ingeniero junior o becario (20000 € anuales).

Teniendo en cuenta las horas laborables totales en un año

$$\left(365 \frac{dias}{año} - \ 129\ dias\ no\ laborables\right) \cdot 8\ \frac{horas\ laborales}{dia} = 1888\ horas\ laborales$$

Estableciendo la relación entre las horas totales y horas de realización del proyecto, se puede obtener un coste aproximado de los costos de la mano de obra

$$Costo\ mano\ de\ obra = \frac{300\ horas\ proyecto}{1888\ horas\ laborables\ anuales} \cdot 20000\ € = \ 3178\ €$$

A este coste hay que sumarle el coste de la cuota patronal (seguridad social), que en el año 2022 está estipulado un sobrecoste de 31.40 % a un 37.05 %, por lo que se tomará la media de estos dos valores, un 34.23 %.

$$Coste\ final\ mano\ de\ obra = 3178\ € \cdot 1.3423 = 4265.83\ €$$

• Los costos de materiales. Aquí se incluyen los costes de los materiales para el desarrollo del proyecto y costes de licencias.

Dado que para el desarrollo del proyecto el software utilizado es de código abierto o se generado mediante programación, el coste de licencias para este proyecto es de 0 euros. En caso de querer ampliar a algún plan de pago (por ejemplo, en la base de datos) o utilizar software distinto que sí que requiera de licencias, habrá que reflejarlo en los costos.

En cuanto a los costes de los materiales para el desarrollo del prototipo, se obtienen realizando el sumatorio del precio de todo el material utilizado para el desarrollo del prototipo.

Materiales utilizados en el desarrollo del prototipo del nodo físico:

- Placa de desarrollo Arduino UNO (ATMega328P): 5 €
- Cable USB 2.0 tipo B: 5 €
- Sensor DHT11: 2.95 €
- Sensor MQ-135: 3.33 €
- Módulo ESP8266 ESP-01: 2.56 €
- Tira de 36 pines (zócalos): 2 €
- Placa de baquelita 1.78 €
- Alambre de soldadura (estaño): 1 €
- Tornillos, tuercas y pasadores: 1 €

  Coste total: 24.62 €

  El resto del sistema es software de código abierto o código generado manualmente, por lo que no se considerará más coste de material para el desarrollo del proyecto.

  Por tanto, obtenemos:

$$Costos\ de\ materiales = 24.62\ €$$

Por lo tanto, los costos directos del proyecto desarrollando un prototipo dan un total:

$$Costos\ directos = Costo\ material + Costo\ mano\ de\ obra = 4290.45\ €$$

Costos indirectos

En cuanto a los costes indirectos, se ha de incluir el coste de todo aquello que ha permitido el desarrollo del proyecto, ya sean costes generales o de producción.

Como indica la UAFSE, para financiar los costes indirectos se aplicará el 15% sobre los costes directos:

$$Costos\ indirectos = 4290.45\ € \cdot \frac{15}{100} = 643.57\ €$$

Costos totales

Finalmente, calcular el costo total del proyecto, será el resultado de sumar los costes directos con los costes indirectos:

$$Costos\ totales = Costos\ directos + Costos\ indirectos = 4934.02\ €$$

# 8 Implicaciones ambientales y sociales

El Internet of Things es una herramienta tecnológica revolucionaria, hasta el punto de ofrecer una nueva forma de entender la tecnología y la vida cotidiana.

En este apartado se hablará las implicaciones que suponen los sistemas IoT en nuestra ciudad inteligente desde un punto de vista social y ambiental.

## 8.1 Implicaciones ambientales

Existen muchas aplicaciones de basadas en sistemas de IoT dedicadas a la correcta conservación del medio ambiente. Estas aplicaciones tratan de regular el impacto humano sobre el entorno, ya sea de forma activa (comunicar con dispositivos para que actúen en consecuencia) o pasiva (notificar de la situación y actuar en función de los datos obtenidos). En el caso de este proyecto, cuyo objetivo final es replicar una red de nodos en la zona geográfica de una ciudad inteligente, se podría clasificar en el segundo grupo, ya que en el punto en el que a fecha de realización de este proyecto, las funciones que posee la interfaz de usuario son de monitorización.

De hecho, existe una vertiente del IoT centrada en sistemas medioambientales llamada Environmental IoT, centrada en conocer tendencias ambientales en el entorno para ejecutar acciones de prevención, en la cual se incluye el desarrollo de proyectos como éste.

El Environmental IoT aplicado en el caso de las Smart Cities, brinda una mejor gestión energética de la ciudad y de la producción industrial, ya que aparte de ser capaces de monitorizar y prevenir, actuando en consecuencia, se pueden agregar dispositivos Smart Grid para ofrecer redes de distribución eléctrica inteligentes al sistema.

Otro ejemplo enfocado a ciudades inteligentes viene dado por los sensores de calidad de aire o calidad de agua, que permitan así localizar sustancias nocivas y actuar en consecuencia, previniendo así patologías o deterioros en sistemas inmunológicos. O mediante los sensores de calidad de aire y agua, controlar las emisiones de $CO_2$ y vertidos tóxicos en ríos que emiten las fábricas en las zonas industrializadas de la ciudad.

Existen muchos otros ejemplos en los cuales los sistemas IoT permiten tomar medidas o actuar de manera automática a favor del medio ambiente en función de los datos que obtienen, los cuales implican la eficiencia de iluminación, mejorar el gasto en calefacción o refrigeración, etc.

Consultando diversos estudios publicados por el IEEE (Instituto de Ingenieros Eléctricos y Electrónicos), aseguran que el desarrollo y crecimiento de dispositivos y sistemas IoT centrados en implicaciones medioambientales, está ofreciendo un gran impacto en la reducción de emisiones de carbono en los países donde se están poniendo en práctica, con una previsión potencial de reducir hasta 63.5 gigatones de $CO_2$ a escala mundial en 2030, reduciendo así la huella de carbono.

Cabe mencionar, que los dispositivos que conforman los nodos del sistema IoT tienen una vida útil, de forma que, una vez queden obsoletas, se debe realizar una recogida para evitar que produzcan un impacto en el entorno.

## 8.2 Implicaciones sociales

Bien es cierto que el concepto de Smart City ha ido evolucionando a lo largo de los años hacia 'una visión más participativa, inclusiva y "urbana"' (March H, 2018).

Cabe mencionar, que el concepto de Smart City supone un cambio tecnológico, dando soluciones tecnológicas a conflictos y situaciones sociales, que no pueden anteponerse al cambio social. Hilando con a los argumentos relacionados con implicaciones medioambientales, donde se brindaba una mayor gestión energética gracias a actuar en relación a sistemas IoT, esto produciría una mayor eficiencia energética con la que cabe la posibilidad, para este caso, que aumente el consumo energético debido al efecto rebote (paradoja de Jevons: la introducción de tecnologías con mayor eficiencia energética pueden aumentar el consumo total de energía).

Otro impacto a evitar, y uno de los motivos por lo que se promueve una visión más participativa, tanto por el sector público como el privado, ya que hay algunos modelos que no regulan sus estrategias de Smart City y terminan siendo excesivamente controladas por el sector privado. Es por ello que el sector público debe interesarse y involucrarse en este tipo de crecimientos tecnológicos, que como ya se ha mencionado a lo largo de este documento, poseen una alta escalabilidad y posee múltiples aplicaciones.

También hay que tener en cuenta el salto tecnológico que este tipo de sistemas implica en la sociedad actual, donde hay una evolución acelerada hacia modelos mayormente tecnológicos, donde las tecnologías IoT e industrias 4.0 cada vez están teniendo más fuerza.

De hecho, existen previsiones por parte de Statista que prevén más 25 mil millones de dispositivos IoT conectados en 2030, crecimiento precedido por una inversión creciente en IoT y un incremento en el gasto mundial en iniciativas que apoyan el desarrollo de Smart Cities, creciente año tras año.

Por lo tanto, existe un interés cada vez mayor en este tipo de sistemas y dispositivos IoT, que avanza cada vez más junto a la revolución tecnológica que supondrá la Industria 4.0, requiriendo todos estos sistemas de gestión de datos, ya sea destinado a Smart Homes, Smart Cities, IoT, IIoT, Smart Gadgets, sistemas Smart Grid y un largo etcétera. Pese a esto, será necesario mentalizar y educar a la población y modernizar los sectores tanto públicos como privados para ofrecer un buen uso y aplicación de este tipo de sistemas, desarrollados para ofrecer solución a problemas que han ido surgiendo debido a la globalización y crecimiento demográfico humano.

# 9 Conclusiones

El trabajo partió de una propuesta en cuanto a ampliación de conocimientos respecto a funcionamiento y arquitectura de microcontroladores y terminó siendo un proyecto en el que se pretende desarrollar un sistema IoT para aplicaciones de gestión en Smart Cities. Para ello, se propuso partir de una placa de desarrollo que estuviera basada en un microcontrolador programable que incorporara el mínimo de periféricos posibles. Para cumplir este fin, las placas Arduino UNO (EE.UU.) / Genuino UNO (Europa) basadas en el microcontrolador ATMega328P, han sido la propuesta como base del desarrollo para el desarrollo del nodo físico de la plataforma IoT a desarrollar.

## 9.1 Objetivos cumplidos

El objetivo principal de este proyecto es el desarrollo de una plataforma IoT, teniendo en cuenta que como requisitos había que utilizar un protocolo IoT para la comunicación entre dispositivos, el desarrollo físico de uno de los nodos en base a la placa de Arduino y que se pudieran visualizar las mediciones a través de una interfaz gráfica.

Se han evaluado distintas alternativas para el desarrollo de la plataforma, de las cuales se ha hecho una selección para el desarrollo del proyecto.

En cuanto a la interfaz de usuario, en este proyecto se ha programado una aplicación móvil en Android vinculada a la base de datos, donde se almacenan los datos obtenidos por la red de nodos. Dado que se ha utilizado un lenguaje de desarrollo de aplicaciones multiplataforma, permite tanto ejecutar el mismo código de la aplicación en sistemas operativos diferentes, como ejecutarla en un navegador web de forma local.

Finalmente, se ha logrado validar el correcto funcionamiento de la plataforma mediante el envío de datos a través del nodo físico, poniendo a prueba el servidor donde tenemos alojado el bróker MQTT, el servidor donde gestionamos los datos a través de servicios y protocolos de transmisión, el almacenamiento de los datos en la base de datos Real-Time Database de Firebase y la lectura de éstos en la interfaz de usuario desarrollada.

## 9.2 Propuestas de futuro

• El proyecto puede seguir escalando, tanto en número de aplicaciones y mediciones distintas, número de nodos, profundidad de desarrollo y evolución de la interfaz.

• Añadir seguridad tanto en los protocolos utilizados como en la base de datos y la aplicación.

• Resultaría interesante desarrollar otras plataformas mediante algunas de las alternativas propuestas y realizar comparaciones entre ellas, para ver hasta qué punto se obtendrían resultados diferentes entre plataformas, de forma cualitativa.

• Implementar el nodo físico desarrollado en una red de nodos real, formando parte de una red de nodos en una Smart City, ya sea para aplicaciones de Environmental IoT o para recolectar datos de puntos geográficos concretos.

# 10 Referencias

**Hardware**

Placa de desarrollo Arduino UNO/Genuino UNO

Arduino Official Store. Arduino UNO R3 Datasheet, *Recursos, documentación Arduino* [en línea]. Enero 2022 p. 2-10. [Última consulta: 28/01/2022]. Arduino® UNO R3, 26/01/2022. Disponible en: https://docs.arduino.cc/resources/datasheets/A000066-datasheet.pdf

Sensor de temperatura y humedad DHT11

Mouser: OSEPP Electronics. DHT11 Humidity & Temperature Sensor. *OSEPP Electronics Product Line* [en línea], p 4-8. [Última consulta: 26/01/2022]. Disponible en: https://www.mouser.com/datasheet/2/758/DHT11-Technical-Data-Sheet-Translated-Version-1143054.pdf

Sensor de calidad de aire MQ-135

Datasheet: HANWEIELCETRONICS CO. LTD. TECHNICAL DATA: MQ-135 GAS SENSOR. *PDF MQ-135 Data sheet (Hoja de datos)* [en línea] p 1-2. [Última consulta 26/01/2022]. Disponible en: http://www.datasheet.es/PDF/605076/MQ-135-pdf.html

Módulo WiFi: ESP8266 ESP-01

Espressif Systems. *ESP8266EX Datasheet* [en línea] p 4-22 (Última actualización: Octubre 2020). [Última consulta: 13/10/2021]. Disponible en: https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf

**Software**

Microchip Studio

Microchip. *Microchip Studio* [Programa: Entorno de programación]. Versión: 7.0. [Última consulta: 31/09/2021]. Disponible en: https://www.microchip.com/en-us/tools-resources/develop/microchip-studio

MQTT Mosquitto Server

Eclipse Foundation. *Eclipse Mosquitto™* [Programa: Protocolo de tranmisión]. Versión: 2.0.14. [Última consulta: 20/01/2022]. Disponible en: https://mosquitto.org/

Node-Red

JS Foundation. *Node-Red: Low-code programming for event-driven applications* [Programa: Interfaz de programación]. Versión: 2.2.2 (npm). [Última consulta: 11/01/2022]. Disponible en: https://nodered.org/

Docker

Docker, Inc. *Docker/Docker Desktop* [Programa]. Versión: 4.3.2 (Última: 4.6.1). [Última consulta: 11/01/2022]. Disponible en: https://www.docker.com/products/docker-desktop/

Node.js

Node js Developers Joyent. *Node.js* [Programa: Digrigido por eventos]. Versión: 14.17.4 [Consulta: 02/12/2021]. Disponible en: https://nodejs.org/es/

Visual Studio Code

Microsoft. *Visual Studio Code* [Programa: Entorno de programación]. Versión: 1.64.2. [Última consulta: 27/01/2022]. Disponible en: https://code.visualstudio.com/

Microsoft Visual Studio

Microsoft. *Microsoft Visual Studio* [Programa: Entorno de programación]. Versión: 2019 [Última consulta: 31/09/2021]. Disponible en: https://visualstudio.microsoft.com/es/

Arduino IDE

Arduino LLC. *Arduino IDE* [Programa: Entorno de programación]. Versión: 1.8.16. [Última consulta: 18/09/2021]. Disponible en: https://www.arduino.cc/en/software

Flutter

Google. *Flutter* [Programa: Kit de desarrollo de software]. Versión: 2.2.3. [Consulta: 10/10/2021]. Disponible: https://api.flutter.dev/

Firebase

Google. *Firebase* [Programa: Plataforma de desarrollo]. Versión: 9.0.5 [Consulta: 24/10/2021]. Disponible: https://firebase.google.com/?hl=es-419

Oracle: *¿Qué es el Internet of Things?* [en línea]. Oracle CloudWorld, 2019. [Consulta: 15/09/2021]. Disponible en: https://www.oracle.com/es/internet-of-things/what-is-iot/

Universitat Carlemany: *Internet de las cosas: definición y ejemplos* [en línea]. Universitat Carlemany, 22 de Junio 2020. [Consulta: 15/09/2021]. Disponible en: https://www.universitatcarlemany.com/actualidad/internet-de-las-cosas-definicion-y-ejemplos

Eduardo W.: *¿Qué es IoT?* [en línea]. Platzi, 2019. [Consulta: 15/09/2021]. Disponible en: https://platzi.com/blog/que-es-iot/?utm_source=google&utm_medium=paid&utm_campaign=14603491644&utm_adgroup=&utm_content=&gclid=CjwKCAiArOgOBhBmEiwAsgeLmewilNFqNi-

PAS1U1CNt2CM0UjZciTPAd4Fe1gTmMR3MTaIr5MEOMhoCqWwQAvD_BwE&gclsrc=aw.ds

Maria G.: *IoT – Internet of Things* [en línea]. Deloitte, 2020. [Consulta: 15/09/2021]. Disponible en: https://www2.deloitte.com/es/es/pages/technology/articles/IoT-internet-of-things.html

Endesa Educa: *Smart Cities* [en línea]. Endesa, 2014. [Consulta: 17/09/2021]. Disponible en: https://www.fundacionendesa.org/es/educacion/endesa-educa/recursos/smart-city

Endesa Educa: *¿Qué es una Smart City?* [Vídeo]. Endesa, 28 de octubre de 2014. [Consulta: 17/09/2021]. Disponible en: https://www.youtube.com/watch?v=lKpoi8lf_tI

Barbara IoT*: Plataformas IoT: que son y cómo pueden beneficiar a tu empresa* [en línea]. Barbara, 16 de abril de 2021. [Consulta: 18/09/2021]. Disponible en: https://barbaraiot.com/blog/plataformas-iot-que-son-y-como-pueden-beneficiar-a-tu-empresa/

Editorial de Geekflare: *12 plataformas y herramientas de Internet de las cosas (IoT) de código abierto* [en línea]. 21 Agosto 2020. [Consulta: 19/09/2021]. Disponible en: https://geekflare.com/es/iot-platform-tools/

Arquitectura del nodo

Programatumicro: *Introducción a IoT* [Vídeo]. Programatumicro, 21 noviembre de 2019. [Consulta: 19/09/2021]. Disponible en: https://www.youtube.com/watch?v=gushPu2dxK4&t=459s

Luis del Valle Hernández: *NodeMCU tutorial paso a paso desde cero* [en línea]. Programarfacil, 20/06/2017. [Consulta: 1/10/2021]. Disponible en: https://programarfacil.com/podcast/nodemcu-tutorial-paso-a-paso/

Thingsboard Documentation: *Temperature upload over MQTT using Raspberry Pi and DHT22 sensor* [en línea]. Thingsboard, 4 Octubre 2017. [Consulta: 02/10/2021]. Disponible en: https://thingsboard.io/docs/samples/raspberry/temperature/

Zach Wendt: *Presentamos las mejores diez placas de desarrollo de 2018* [en línea]. Arrow, 22 Marzo 2018. [Consulta: 04/10/2021]. Disponible en: https://www.arrow.com/es-mx/research-and-events/videos/the-top-10-development-platforms-dev-kits-2018

Silicon Labs: *Thunderboard™ Sense 2: Kit de desarrollo de Internet de próxima generación* [en línea]. Digi-Key, 03/11/2017. [Consulta: 07/10/2021]. Disponible en: https://www.digikey.es/es/product-highlight/s/silicon-laboratories/thunderboard-sense-2-next-generation-iot-development-kit?utm_adgroup=General&utm_source=google&utm_medium=cpc&utm_campaign=Dynamic%20Search_ES_Product&utm_term=&productid=&gclid=CjwKCAjwu_mSBhAYEiwA5BBmf2f9Mnt3nCo_MAmBNWugFp7rUzdD_vpp03ElnfE9zxIYLQ77aBFiyRoCn9QQAvD_BwE

ST life augmented: *STM32 32-bit Arm Cortex MCUs* [en línea]. ST, Junio 2007. [Consulta: 07/10/2021]. Disponible en: https://www.st.com/en/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus.html#documentation

Redacción de Naylamp Mechatronics: *Usando ESP8266 con el IDE de Arduino* [en línea]. Naylamp Mechatronics, 14/11/2016. Disponible en: https://naylampmechatronics.com/blog/56_usando-esp8266-con-el-ide-de-arduino.html

Dr. Fernando Moutinho Deyán: *IoT (Internet Of Things). Internet de las cosas. ESP8266-01. Guía Vol: 1: Descripción, funcionamiento, configuración y proyectos* [en línea]. Dr. Fernando Moutinho Deyán, 2018. Disponible en: https://www.amazon.co.uk/Internet-Things-cosas-ESP8266-01-Gu%C3%ADa/dp/1726722627/ref=sr_1_17?s=books&ie=UTF8&qid=1551929469&sr=1-17&keywords=esp8266

Mobizt. *Firebase-ESP8266* [Programa: Librería de programación]. Versión: 3.6.5, 22 Noviembre 2021 [Consulta: 24/11/2021]. Disponible: https://github.com/mobizt/Firebase-ESP8266

Protocolos IoT

Barbara IoT: *Protocolos IoT que deberías conocer* [en línea]. Barbara, 29 de abril de 2021. [Consulta: 18/09/2021]. Disponible en: https://barbaraiot.com/blog/protocolos-iot-que-deberias-conocer/

Pick Data: *MQTT vs CoAP, la batalla por ser el mejor protocolo IoT* [en línea]. PickData, 21 Oct 2019. [Consulta: 19/09/2021]. Disponible en: https://www.pickdata.net/es/noticias/mqtt-vs-coap-mejor-protocolo-iot

Base de datos

Google Firebase: *Integraciones en Cloud Firestore: Usa Cloud Firestore con Firebase Realtime Database* [en línea]. Firebase, 18/11/2021. [Consulta: 21/11/2021]. Disponible en: https://firebase.google.com/docs/firestore/firestore-for-rtdb?hl=es-419

Amazon Web Services: *Amazon DynamoDB* [en línea]. Amazon. [Consulta: 22/11/2021]. Disponible en: https://aws.amazon.com/es/dynamodb/?p=ft&c=db&z=3

MongoDB: *ATLAS Database. Deploy a multi-clout database* [en línea]. MongoDB. [Consulta: 22/11/2021]. Disponible en; https://www.mongodb.com/atlas/database

Freddier: *Curso de Fundamentos de Bases de Datos – Qué es SQL y NoSQL* [en línea]. Freddier, Platzi, 2018. [Consulta: 22/11/2021] Disponible en: https://platzi.com/blog/sql-nosql/?utm_source=google&utm_medium=paid&utm_campaign=14603491644&utm_adgroup=&utm_content=&gclid=Cj0KCQjwl7qSBhD-ARIsACvV1X3pxqHBCyGvls-CdyWnFK-fo6J1wtfxCoOHYaoYIrXgZ9z4tRpXsi4aApVGEALw_wcB&gclsrc=aw.ds

Interfaz de usuario

Rubén Velasco: *¿Quieres hacer apps para móviles? Estos son los lenguajes que debes conocer* [en línea]. Rubén Velasco, SoftZone, 10 Agosto 2021. [Consulta: 30/11/2021]. Disponible en: https://www.softzone.es/programas/lenguajes/programar-apps-moviles/

Immune: *Lenguajes de programación para móvil* [en línea]. Immune: Technology Institute, 14/10/2021. [Consulta: 30/11/2021]. Disponible en: https://immune.institute/lenguajes-de-programacion-para-movil/

Redacción KeepCoding: *Los lenguajes de programación para aplicaciones móviles más usados* [en línea]. KeepCoding Tech School, 25/01/2022. [Consulta: 20/02/2022]. Disponible en; https://keepcoding.io/blog/lenguajes-de-programacion-para-aplicaciones/

*Usando el ATmega328 como una alternativa a Arduino*. Wikipedia [en línea]. Wikimedia Foundation, 2021. [Consulta: 20/09/2021]. Disponible en: https://es.wikipedia.org/wiki/Atmega328

Newton C. Braga: *Conociendo el microcontrolador núcleo (Core) ATmega328P de Arduino Uno. (MIC019S)* [en línea]. Newton C. Braga, Instituto NCB. [Consulta 20/09/2021]. Disponible en: http://www.incb.com.mx/index.php/articulos/78-microcontroladores-y-dsps/2546-conociendo-el-microcontrolador-nucleo-core-atmega328p-de-arduino-uno-mic019s

Adaruit. *Adafruit Unified Sensor Driver* [Programa: Librería]. Versión: 1.1.4. [Última consulta: 09/10/2021]. Disponible en: https://github.com/adafruit/Adafruit_Sensor

Luis del Valle Hernández: *ESP8266 todo lo que necesitas saber del módulo WiFi para Arduino* [en línea]. Luis del Valle Hernández, Programarfacil, 2016. [Última consulta: 27/01/2022]. Disponible en: https://programarfacil.com/podcast/esp8266-wifi-coste-arduino/

Espressif: *ESP8266_RTOS_SDK* [Programa: Librería]. Versión 3.4. [Última consulta: 28/01/2022]. Disponible en: https://github.com/espressif/ESP8266_RTOS_SDK

Oscar Fernandez Alzate: *Monitor de temperatura web con ESP8266 y Arduino* [en línea]. Oscar Fernandez Alzate, CódigoElectrónica, 16 enero 2020. [Última consulta: 10/01/2022]. Disponible en: http://codigoelectronica.com/blog/monitor-de-temperatura-web-con-esp8266-y-arduino#codigo

Oscar Fernandez Alzate: *ESP8266 Flasher Firmware* [en línea]. Oscar Fernandez Alzate, CódigoElectrónica, 25 noviembre 2019. [Última consulta: 10/01/2022]. Disponible en: http://codigoelectronica.com/blog/esp8266-flasher-firmware

Oscar Fernandez Alzate: *Conexión MQTT con Arduino* [en línea]. Oscar Fernandez Alzate, CódigoElectrónica, 29 noviembre 2019. [Última consulta: 10/01/2022]. Disponible en: http://codigoelectronica.com/blog/esp8266-conexion-a-mqtt-con-arduino

Andrew Shvayka, Igor Kulikov, Igor Khanenko: *Temperature Dashboard Using Arduino UNO, ESP8266 And MQTT* [en línea]. Team Thingsboard, 12 Diciembre 2016. [Última consulta: 10/01/2022]. Disponible en: https://create.arduino.cc/projecthub/thingsboard/temperature-dashboard-using-arduino-uno-esp8266-and-mqtt-5e26eb

MQTT: *MQTT: The Standard for IoT Messaging* [en línea]. MQTT. [Consulta: 18/01/2022]. Disponible en: https://mqtt.org/

HiveMQ: *MQTT Essentials* [en línea]. HiveMQ. [Consulta: 18/01/2022]. Disponible en: https://www.hivemq.com/mqtt-essentials/

Steve Cope, Steve's Internet Guide: *Practical MQTT with Steve* [en línea]. Steve's Internet Guide, 2021. [Consulta: 18/01/2022]. Disponible en: http://www.steves-internet-guide.com/mosquitto-broker/

Redacción de Eclipse Mosquitto: *The Server* [en línea]. Eclipse Mosquitto. [Última consulta: 20/01/2022. Disponible en: https://test.mosquitto.org/

Redacción de Eclipse Mosquitto: *Version 2.0.0 released* [en línea]. Eclipse Mosquitto, 03/12/2020. [Última consulta: 20/01/2022. Disponible en: https://mosquitto.org/blog/2020/12/version-2-0-0-released/

Redacción de Eclipse Mosquitto: *Migrating from 1.x to 2.0* [en línea]. Eclipse Mosquitto, [Última consulta: 20/01/2022]. Disponible en: https://mosquitto.org/blog/2020/12/version-2-0-0-released/

Nick O'Leary: *Version 2.1 released* [en línea]. Nick O'Leary, Node-RED, 21 Octubre 2021. [Consulta: 27/10/2021]. Disponible en: https://nodered.org/blog/2021/10/21/version-2-1-released

JohanIoT: *Getting data from weather API* [en línea]. JohanIoT, Node-RED, Noviembre 2019. [Última consulta: 14/11/2021]. Disponible en: https://flows.nodered.org/flow/4d9cfc53b57640924656b8687c0b8d3c

Thomas Südbröcker: *Absolute beginner with Node-RED: (once more) using HTTP request to get data from a REST endpoint* [Vídeo, en línea]. Thomas Südbröcker, YouTube, 6 julio 2021. [Última consulta: 14/11/2021]. Disponible en: https://www.youtube.com/watch?v=I_ulT4xRoTA

ElectroSystem: *Subir datos de Arduino UNO a Firebase con Node-Red sin módulo WiFi* [Vídeo, en línea]. ElectroSystem, 3 Septiembre 2021. [Consulta: 15/11/2021]. Disponible en: https://www.youtube.com/watch?v=DQU9MGUlteQ

Steve Cope: *Node-Red HTTP Request Node Posting Data to Thingsboard* [Vídeo, en línea]. Steve Cope, YouTube, 14 Diciembre 2018. [Consulta: 14/11/2021]. Disponible en: https://www.youtube.com/watch?v=wW_UegeszEs

Steve Cope: *How to Log MQTT Sensor Data in Node-Red* [Vídeo, en línea]. Steve Cope, YouTube, 22 Mayo 2018. [Consulta: 14/11/2021]. Disponible en: https://www.youtube.com/watch?v=oo6AjJia6ko

Agencia Estatal de Meteorología: *AEMET OpenData: Sistema para la difusión y reutilización de la información de AEMET* [en línea]. AEMET. [Consulta: 30/09/2021]. Disponible en: https://opendata.aemet.es/centrodedescargas/inicio

Visual Crossing: *Visual Crossing Weather: Weather Data & API* [en línea]. Visual Crossing. [Consulta: 15/11/2021]. Disponible en: https://www.visualcrossing.com/

Josefina López Herrera. Diseño de sistemas de tiempo real. *Programación en tiempo real y bases de datos: un enfoque práctico*. Primera edición: Septiembre de 2011. Oficina de Publicacions Acadèmiques Digitals de la UPC, 2011, p. 19-34. ISBN: 978-84-7653-686-5. Disponible en: https://books.google.es/books/about/Programaci%C3%B3n_en_tiempo_real_y_bases_de.html?id=fvdoBQAAQBAJ&printsec=frontcover&source=kp_read_button&hl=es&redir_esc=y#v=snippet&q=id&f=false

Firebase: *The Firebase Realtime Database and Flutter – Firecasts* [Vídeo, en línea]. Firebase, 19 Julio 2021. [Consulta: 03/12/2021]. Disponible en: https://www.youtube.com/watch?v=sXBJZD0fBa4&list=LL&index=21

Google Firebase, FlutterFire. Get Started. Read and Write Data. Work with Lists of Data. *Realtime Database* [en línea]. FlutterFire. [Consulta: 03/12/2021]. Disponible en: https://goo.gle/3ziSc1x

Firebase: *Introducción a las reglas de seguridad de Firebase - Firecasts* [Vídeo, en línea]. Firebase, 19 Julio 2019. [Consulta: 19/12/2021]. Disponible en: https://www.youtube.com/watch?v=QEuu9X9L-MU

Firebase: *Explicación de las reglas de Firebase Realtime Database*. Seguridad y reglas. Comprender las Reglas [en línea]. Firebase, 1 Octubre 2021. [Consulta: 19/12/2021]. Disponible en: https://firebase.google.com/docs/database/security

Pau Fernández Durán (Pauek): *Flutter* [Vídeos, en línea]. Pau Fernández, 2 Agosto 2019. [Consulta: 21/09/2021]. Disponible en: https://www.youtube.com/watch?v=kEPCVbpz70g&t=456s

Redacción IThink UPC: *Recomendaciones de seguridad en dispositivos IoT* [en línea]. IThink UPC. [Consulta: 03/03/2022]. Disponible en: https://www.ithinkupc.com/es/blog/recomendaciones-de-seguridad-en-dispositivos-iot

Eclipse Mosquitto: *Authentifcation methods* [en línea]. Eclipse Mosquitto. [Última consulta: 03/03/2022]. Disponible en: https://mosquitto.org/documentation/authentication-methods/

Redacción Glassdoor: *Sueldos para Ingeniero Junior* [en línea]. Glassdoor, Marzo 2022. [Consulta: 11/03/2022]. Disponible en: https://www.glassdoor.es/Sueldos/ingeniero-junior-sueldo-SRCH_KO0,16.htm

Redacción Sesame: *Calculadora de jornada de trabajo* [en línea]. Sesame, 2015. [Consulta: 11/03/2022]. Disponible en: https://www.sesametime.com/assets/calculadora-de-jornada-de-trabajo/

Alejandro Aradas: *Porcentaje de cotización a la Seguridad Social del trabajador y empresa* [en línea]. Alejandro Aradas, Cuestiones Laborales, 1 Enero 2022. Disponible en: https://www.cuestioneslaborales.es/porcentaje-de-cotizacion-a-la-seguridad-social-del-trabajador-y-empresa/

Unidad Administradora del Fondo Social Europeo: *Consulta aplicación 15% para el cálculo de los costes indirectos en las ayudas a la contratación* [en línea]. UAFSE, 2020. [Consulta: 11/03/2022]. Disponible en: https://www.mites.gob.es/uafse/es/simplificacion/consultas/respuestas/resp_costesindirectos_rci.htm

T. P. Mendieta, J. F. Herrera, A. L. Jiménez (2019): *La Capacidad del IoT de Transformar el Futuro* [Dataset]. Fundación Avenir. Disponible en: https://fundacionavenir.net/revista/index.php/avenir/article/download/79/27

Alicia Asín Pérez: *Las oportunidades del IoT y el control del impacto ambiental en el sector de la edificación* [en línea]. Alicia Asín Pérez, Directora Gerente, Libelium, 27/09/2018. [Consulta: 21/02/2022]. Disponible en: https://www.casadomo.com/comunicaciones/comunicacion-oportunidades-iot-control-impacto-ambiental-sector-edificacion

Redacción EUDE Business School: *El Internet de las cosas y su influencia en el medio ambiente* [en línea]. EUDE Business School, 09 Enero 2019. [Consultado en: 21/02/2022]. Disponible en: https://www.eude.es/blog/internet-medio-ambiente/

Redacción Zemsania Global Group: *Environmental IoT: ¿Cómo puede la tecnología mejorar nuestro entorno?* [en línea]. Zemsania Global Group. [Consulta: 21/02/2022]. Disponible en: https://zemsaniaglobalgroup.com/environmental-iot-mejorar-entorno/

Gerard George, Simon J. D. Schillebeeckx: *Digital sustainability and its implications for finance and climate change* [en línea]. Gerard George, Simon J. D. Schillebeeckx, Singapore Management University, Abril 2021. [Consultado en: 21/02/2022]. Disponible en: https://ink.library.smu.edu.sg/cgi/viewcontent.cgi?article=7720&context=lkcsb_research

Redacción Cloud.Studio: *Huella de Carbono e IoT* [en línea]. Cloud.Studio, 8 Marzo 2021. [Consulta: 21/02/2022]. Disponible en: https://www.cloud.studio/2021/03/08/huella-de-carbono-e-iot/

Redacción de Universia: *El internet de las Cosas y su impacto transformador en la sociedad*. Universia, 27 Julio 2021. [Consulta: 23/02/2022]. Disponible en: https://www.universia.net/es/actualidad/orientacion-academica/el-internet-de-las-cosas-y-su-impacto-transformador-en-la-sociedad.html

Hug March Corbella: *The right to the (smart) city?: El giro social del paradigma de Smart Cities o Ciudades Inteligentes* [en línea]. Hug March Corbella, UOC, 17 Diciembre 2018. [Consulta: 23/02/2022]. Disponible en: https://blogs.uoc.edu/economia-empresa/es/the-right-to-the-smart-city-el-giro-social-del-paradigma-de-smart-cities-o-ciudades-inteligentes/

Wikipedia: *Paradoja de Jevons* [en línea]. Wikimedia Foundation, 2022. [Consulta: 23/02/2022]. Disponible en: https://es.wikipedia.org/wiki/Paradoja_de_Jevons

Rosa Fernández: *El Internet de las cosas (IoT) – Datos estadísticos* [en línea]. Rosa Fernández, Statista, 24 Marzo 2022. [Consulta: 27/03/2022]. Disponible en: https://es.statista.com/temas/6976/el-internet-de-las-cosas-iot/#topicHeader__wrapper

# 11 Anexos

## Anexo A. Listado de herramientas utilizadas en el desarrollo del trabajo

Herramientas de hardware:

- Arduino UNO / Genuino UNO – Microcontrolador: ATMega328P: Placa de desarrollo que incluye el microcontrolador programable.
- DHT11: Sensor digital de temperatura y humedad con el que recolectaremos datos en el nodo físico.
- MQ-135: Sensor de gas para control de calidad de aire, admite detección de NH3, NOx, alcohol, benceno, amoníaco y CO2, entre otros.
- Cableado (Evaluación de soldadura final)
- ESP8266 ESP-01: Módulo Wi-Fi integrado que contiene un microcontrolador de rendimiento confiable para desarrollos en internet de las cosas.

Herramientas para desarrollo de software:

- Microchip Studio: Para primeras pruebas de funcionamiento de la placa, siendo éste un entorno de programación más completo para el desarrollo y depuración de microcontroladores AVR.
- MQTT Eclipse Mosquitto Software: Broker de mensajería que implementa el protocolo MQTT.
- Node-Red: herramienta de desarrollo que, mediante programación visual, permite establecer relaciones y funciones al usuario entre nodos que ejecutan funciones propias de servicios Net Code sin necesidad de programarlas una a una.
- Docker (Docker Desktop): Herramienta que permite ejecutar aplicaciones dentro de contenedores de software.
- Visual Studio Code: Entorno de desarrollo compatible con la mayoría de lenguajes de programación.
- Visual Studio 2019 (Community Edition): Entorno de desarrollo que incluye diversas funcionalidades para desarrollos en web y APIs.
- Arduino IDE.
- Flutter (Dart): Kit de desarrollo de software basado en Dart como lenguaje de programación, desarrollado por Google, que permite desarrollar interfaces de usuario para aplicaciones Android, iOS y navegador Web.
- Firebase: Plataforma orientada al desarrollo de aplicaciones web y móviles, ofreciendo bases de datos en tiempo real como Firestore y Real-Time Database.
- Node.js: Entorno de ejecución para JavaScript, que nos ofrecerá acceso al servicio "*npm*"

Herramientas auxiliares:

- USB Type-B: Para realizar la descarga del programa al microcontrolador.
- Protoboard: Utilizada en las pruebas de funcionamiento del nodo físico antes de soldar los componentes en un circuito impreso.
- PC (CPU)
- Conector jack con fuente de alimentación o adaptador de 5V (opcional, ya que se puede seguir alimentando mediante USB)

- Microsoft Word 2007 o superior.
- Navegador de internet (Mozilla Firefox, Google Chrome, Internet Explorer o Microsoft Edge)
- Teléfono móvil: Una vez establecida la conexión al servidor de la base de datos para aplicaciones Android, será necesario compilar el código de la aplicación en el teléfono móvil para que permita la depuración.
- Módem WiFi: En caso de utilizar un ordenador portátil o una placa de desarrollo con WiFi incorporado (ejemplo: Raspberry), no es necesario.

## Anexo B. Soluciones alternativas

### Soluciones alternativas respecto a la arquitectura del nodo

#### Utilizar una placa de desarrollo NodeMCU

Este apartado incluye: Wemos D1 Mini, NodeMCU, ESP-12F y LinkNode

La placa de desarrollo NodeMCU es la placa de desarrollo que incorpora un microcontrolador más extendida para desarrollos de IoT, ya que es una placa que incorpora el módulo ESP8266 (MOD ESP-12E) por lo que, es una placa de desarrollo que de entrada, nos ofrece la posibilidad de realizar conexiones inalámbricas a puntos WiFi.

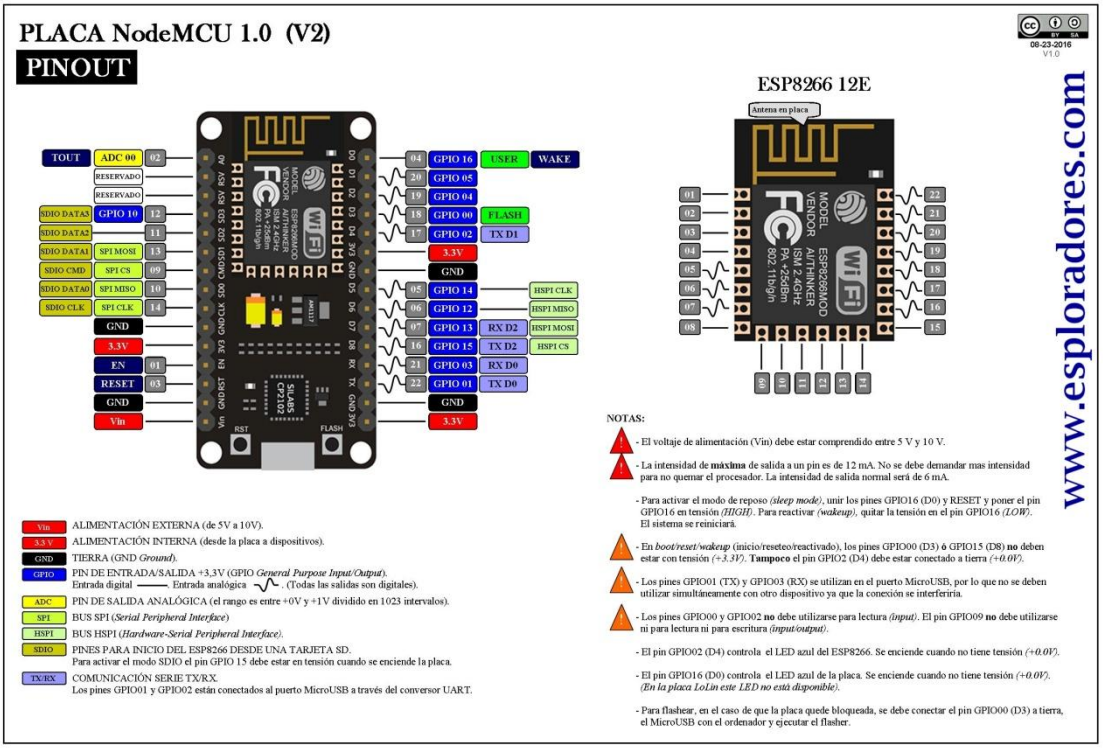


*Figura 40*: Dani N. (2016). [Pines de la placa de desarrollo NodeMCU (Módulo ESP8266 12E)]. ESPloradores. URL: https://www.esploradores.com/wp-content/uploads/2019/11/PINOUT-NodeMCU_1.0-V2-2_2.jpg

Otro de los motivos por los que está tan extendido es la compatibilidad que ofrece con los entornos de Arduino, ya que no solo permitirá reprogramar el microcontrolador mediante los comandos AT que posee el firmware, sino que al ser compatible con entornos de Arduino, se podrá reprogramar mediante los mismos lenguajes y entornos de desarrollo

(cabe destacar que el ESP8266 o ESP-01 no son microcontroladores AVR, pertenecen a Espressif).

También cabe mencionar que las placas NodeMCU son económicamente muy asequibles, hasta el punto de llegar a ser más versátiles cuanto a precios que placas de desarrollo que no poseen ningún tipo de módulo integrado.

Para el caso de este proyecto en específico, el uso de este tipo de dispositivos, resulta algo interesante, ya que posee una librería C/C++ que sigue actualizada hasta día de hoy, compatible con este tipo de dispositivos, llamada Firebase-ESP8266. Esta librería permite realizar una conexión hacia una API de Firebase Real-Time Database, la cual nos permite tanto leer como escribir en la dirección de la base de datos de forma muy simple.

**Utilizar una base Raspberry Pi**

Para este tipo de proyectos esta opción es quizá la que más versatilidad ofrece. Esto es debido a que la Raspberry Pi es una placa de desarrollo basada en Linux que actúa como un computador en miniatura, además ofrece 40 pines de entrada y salida de propósito general. El sistema operativo de esta computadora en miniatura es Raspbian (base Linux) y, gracias a esto y que posee Bluetooth y WiFi también se puede utilizar para administrar los servidores de la red de nodos, además de poseer distintos periféricos añadidos.

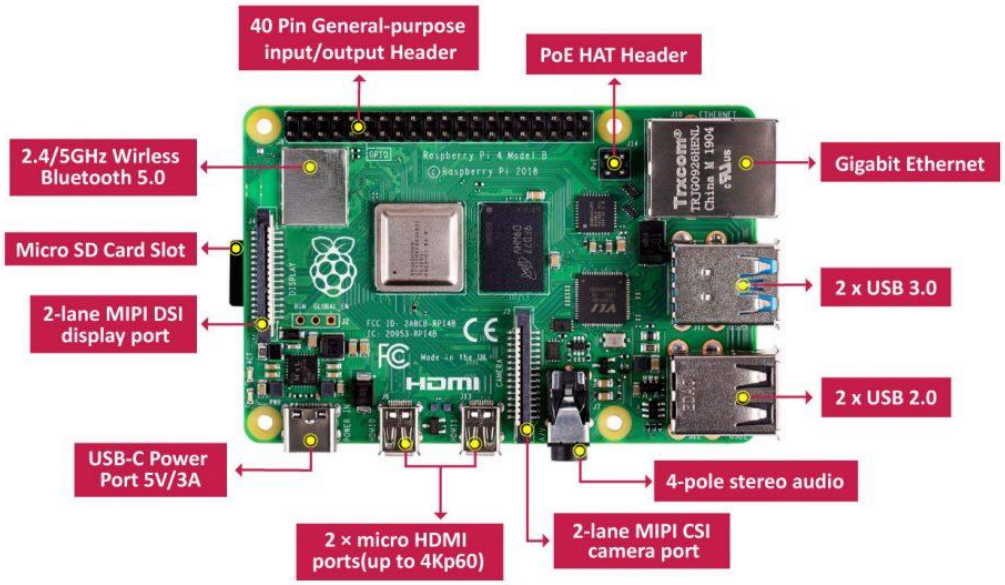


*Figura 41:* ArduinoQue – Robótica (2021). [Raspberry Pi 2, esquema de periféricos]. ArduinoQue. URL: https://arduinoque.com/robots/home-assistant-raspberry-pi-2/

Cabe mencionar, pese a que los dispositivos desarrollados en base Raspberry Pi para desarrollos de IoT están muy extendidos, cada año están aumentando más de valor económico, hasta el punto de haber cuadruplicado o más su precio de venta inicial. Es cierto que existen modelos más económicos, como son las series Zero, pero conllevará a una disminución de las prestaciones y velocidad de procesamiento, que pueden afectar a la escalabilidad del proyecto. Pese a esto, para el tipo de proyecto a realizar, sí que se podría realizar en base a Raspberry Pi Zero W, siempre que sea a nivel de prototipo, en caso de que se quisiera realizar a escala real, con una red de nodos escalable,

necesitaríamos utilizar o bien, distintos dispositivos sincronizados o utilizar dispositivos de procesamiento más potentes (por ejemplo: Raspberry Pi 3).

| | Raspberry Pi 3  | Raspberry Pi Zero W  |
|---|---|---|
| Price | $35 | $10 |
| Dimension | Height: 2.22"<br>Width: 3.37"<br>Weight: 45g | Height: 1.18"<br>Width: 2.55"<br>Weight: 9g |
| SoC | Broadcom BCM2837 | Broadcom BCM2835 |
| Core | 64-bit ARM Cortex-A53 Quad-Core | 32-bit ARM1176JZF-S Single-Core |
| GPU | Broadcom VideoCore IV | Broadcom VideoCore IV |
| CPU Clock | 1.2GHz | 1GHz |
| Memory | 1GB | 512MB |
| SPI/I2C | Yes | Yes |
| Supported OS | Linux, Android Things, Windows 10 IoT Core | Linux |
| Camera Interface | Supported | Supported |
| LCD Panel | Supported | Not Supported |
| GPIO | 40 pins populated | 40 pins unpopulated |
| USB Ports | 4x USB 2.0 | 1x Micro-USB |
| Ethernet | 10/100 Mbit/s | None |
| Wi-Fi/Bluetooth | 2.4GHz 802.11n<br>Bluetooth Classic 4.1 | 2.4GHz 802.11n<br>Bluetooth Classic 4.1 |
| HDMI | HDMI rev 1.3<br>1080p HD Video | Mini-HDMI<br>1080p HD Video |
| Analog Video | Shared with audio jack | Accessed by unpopulated pin |
| Power Rating | 180mA | 1.24A @ 5V |

*Tabla 3:* Gudino, M. (2018). [Comparativa Raspberry Pi 3 vs Raspberry Pi Zero W]. Arrow, URL: https://www.arrow.com/es-mx/research-and-events/articles/raspberry-pi-3-vs-raspberry-pi-zero-w

Dada la utilidad que posee este dispositivo, al ser un ordenador en miniatura, puede ofrecer distintos usos para un proyecto de desarrollo de una red de nodos:
- Utilizar el dispositivo como un nodo más con conexión a internet, añadiendo los periféricos (sensores) en los pines de propósito general. A partir de este punto, se puede optar a utilizar un protocolo de comunicación (como MQTT, por ejemplo) en un servidor de forma local, suscribiendo los datos obtenidos mediante los sensores y un cliente al que se deseen enviar, para poseer una herramienta IoT con la que mover los datos hacia otro servicio.

- Que el dispositivo realice la función de intermediario entre nodos. Muchos de los proyectos de IoT y domótica casera, utilizan este sistema, donde poseen una red de nodos que comunican con un nodo central que sirve de intermediario para sincronizar, transmitir e incluso almacenar (recordar que uno de los periféricos es una memoria SD) los datos obtenidos a partir de la red de nodos. De esta manera, se facilita mucho la transmisión de información de manera bilateral, suscribiendo dispositivos cliente al servidor alojado en la Raspberry, pudiendo no solo recibir datos, sino también,

enviar datos hacia los nodos, facilitando así la conexión entre dispositivos, sería una opción sencilla en caso de que querer realizar un sistema de comunicación bilateral.

**Otras placas de desarrollo**

Para el desarrollo de este proyecto, la arquitectura del nodo se ha realizado en base a la placa Arduino UNO con un microcontrolador ATMega328P debido a la alta compatibilidad que posee con distintos módulos, gracias a la compatibilidad con el microcontrolador, o la capacidad (en caso de que sea necesario) de poder utilizar un escudo de desarrollo con el que ampliar las capacidades de la placa. Pero existen dispositivos especializados en la realización de proyectos de esta índole y enfocados al desarrollo IoT. Siguiendo la recomendación de Arrow Electronics, algunas de las alternativas más recomendables para estos tipos de desarrollos son:

- Para desarrollos similares al del proyecto, cabe dar mención honorífica a la placa de desarrollo Thunderboard Sense 2 de Silicon Labs, placa que incluye una serie de sensores ya integrados y implementación directa de estos sensores con la nube. Es una placa muy flexible y adaptable a distintos proyectos de índole parecida a este, basada en el microcontrolador ARM Cortex-M4 y, además, incluye un módulo de conexión a internet.

- El kit de desarrollo SensorTile de STMicroelectronics, está diseñado principalmente para el desarrollo de aplicaciones de IoT de baja potencia. Es un kit centrado en la placa de desarrollo basada en el módulo STM32 (microcontrolador de 32 bits basado en el procesador ARM Cortex-M) junto con una placa de expansión que brinda de distintas capacidades extra de conexión y desarrollo.

- También mencionar la familia ARTIKA, como el kit de desarrollo ARTIK 710, de Samsung, diseñado específicamente para aplicaciones IoT. El kit también está basado en un módulo ARM Cortex, que ofrece una combinación de WiFi y Bluetooth. Cabe destacar que este kit brinda la posibilidad de añadir cámara, audio y socket para memoria SD.

Existen muchas otras placas y kits de desarrollo para aplicaciones IoT, en este apartado se han comentado algunos dispositivos interesantes que puedan funcionar como alternativas para el proyecto desarrollado, pero aún existen muchas otras alternativas más.

Cabe mencionar que, las placas y kits de desarrollo mencionadas en este apartado, en comparación con NodeMCU y la placa de desarrollo Arduino UNO, poseen precios mucho más elevados a día de la redacción de este archivo, pero ofrecen mayores prestaciones.

## Soluciones alternativas respecto a los protocolos IoT

### Protocolo de acceso de red

En el desarrollo de este proyecto se ha utilizado como protocolo de acceso de red el uso de redes WiFi, debido a que la idea inicial era que los nodos fueran capaces de funcionar de forma independiente y de manera inalámbrica.

Al incorporar el módulo WiFi ESP8266 al proyecto para cumplir esta función en el hardware y programar servidores con protocolos de transmisión bajo modelo TCP/IP funcionando a través de una IP local, además de poseer como punto de acceso a Internet un repetidor con antena WiFi conectado a la red local, junto con los servidores a través de un módem inalámbrico. Por estas razones, el protocolo de acceso a la red ideal para este proyecto, bajo estas condiciones, es via WiFi.

Otras opciones pueden ser Ethernet, 3G y 5G.

Una opción que queda fuera del rango de la idea del proyecto, pero algunos proyectos donde se trata la transmisión de datos mediante placas Arduino. Consiste en utilizar la transmisión serial del puerto USB que contienen las placas de desarrollo. Mediante un script en cualquier lenguaje de programación que permita leer la entrada serial del puerto USB, podemos extraer los datos del nodo mediante el puerto USB y, utilizando la consola que ejecuta el script, podemos almacenar los datos en el propio dispositivo (si lo permite) o suscribir los datos a un protocolo de transmisión IoT para su posterior gestión.

### Protocolo de tranmisión

En el desarrollo de este proyecto se ha utilizado protocolos IT, que sirven para transmitir información a internet. Los protocolos utilizados han sido el protocolo MQTT y protocolos HTTP. El protocolo MQTT ha sido utilizado como servidor bróker en el que se publican los datos obtenidos por parte del nodo y, como cliente, el servidor Node-Red donde se realizaba la gestión.

Los protocolos HTTP han sido utilizados para realizar peticiones a servicios relacionados con bases de datos. Extraer los datos de la web API de Visual Crossing para la obtención de los datos de los nodos remotos para su posterior gestión en el servidor Node-Red y crear nuevos recursos en colecciones de datos en la Real-Time Database de Firebase.

La alternativa al protocolo de transmisión utilizado encaja muy bien con la toma de decisiones del proyecto debido a la naturaleza del protocolo, se trata del protocolo CoAP (Constrained Application Protocol). CoAP es un protocolo cliente-servidor que permite enviar paquetes entre nodos, estos paquetes son evaluados por el servidor y, en función de la lógica del paquete, decidirá qué acción realizar. Este protocolo IoT tiene base UDP, el modelo utilizado es distinto al más extendido (MQTT), pero emplea el modelo REST HTTP y ofrece menor consumo de energía, además ser capaz de gestionar mensajes síncronos. El principal problema por la cual este protocolo no ha sido utilizado en la realización de este proyecto es la falta de soporte por las herramientas utilizadas en comparación con MQTT, el cual ya venía con soporte de serie, por lo que resulta más fácil de implementar y hay más información al respecto.

## Soluciones alternativas respecto a la base de datos

Para almacenar los datos obtenidos por los nodos, en este proyecto se ha utilizado la Real-Time Database de Firebase, ya que esta base de datos ofrece prestaciones útiles comentadas en el apartado en el que se habla en el modelo de la base de datos.

Algunas opciones serían utilizar otros tipos de bases de datos NoSQL como MongoDB Atlas, DynamoDB o Firebase Firestore.

- MongoDB es una solución interesante, ya que ofrece servicios de movilidad a través de AWS (dominio de DynamoDB), Azure (Microsoft), Google Cloud, ofreciendo también una API admitiendo una estructura de datos similar a la planteada en Real-Time Database. Esta base de datos era la segunda opción a barajar, ya que era muy similar al entorno de Firebase, pero la decisión se tomó en cuanto a que, desde el punto de la base de datos hasta la interfaz de usuario, el desarrollo era mejor realizarlo desde el entorno de Google.
- DynamoDB es la solución que nos ofrece el dominio de Amazon, esta sin duda es la que ofrece mayor capacidad de escalar al contenido del proyecto (sin tener que crear diversas instancias) de forma tanto vertical como horizontal. También ofrece un servicio con disponibilidad de datos crítica, junto con servicios y herramientas del dominio (de cara a aplicaciones y análisis de datos).
- Firebase Firestore, una de las ideas era utilizar Cloud Firestore para mover los datos de Firebase Real-Time Database hacia Firebase Firestore, ya que es la primera y única base de datos que ha utilizado el autor antes del inicio del proyecto. Migrar los datos a esta base de datos ofrece una mayor disponibilidad (para un proyecto académico, no es crítico el 99.995% de disponibilidad que ofrece Real-Time Database), permite un mayor flujo de actualizaciones y escrituras en la base de datos y crear colecciones de datos. Utilizar esta base de datos hubiera brindado algunos problemas, como que el número de operaciones está limitado a 10.000 operaciones por día (incluye operaciones de lectura) de forma gratuita de forma que, a la hora de depurar el código, desarrollando una aplicación destinada a ser usada como interfaz de usuario de cientos o miles de mediciones simultáneas, hubiese sido un problema. Si se busca que el proyecto sea escalable, no es la base de datos adecuada, ya que en caso de necesitar agregar nuevas bases de datos, a diferencia de Real-Time Database, esta no posee este servicio para un mismo proyecto.

Otra opción sería usar bases de datos SQL, únicamente en el caso de que la red de nodos, todos los nodos sean del mismo tipo y realicen el mismo tipo de medición, ya que si no, se generaría redundancia. Las bases de datos más típicas para este tipo de desarrollos son MySQL, Oracle y PostgreSQL. Cabe mencionar que la mayoría de desarrollos en IoT, suelen utilizar bases de datos NoSQL, dado que los datos que se suelen almacenar, suelen venir de distintos tipos de sensores que realicen distintos tipos de medición. De esta forma, utilizando una base de datos NoSQL, no hay que preocuparse de la redundancia (ya que es una base de datos no relacional), los datos se transmiten de forma más liviana y a mayor velocidad.

## Soluciones alternativas respecto a la interfaz de usuario

En este apartado, se abren soluciones alternativas respecto al entorno o lenguaje de programación para el desarrollo de la plataforma e infinitas soluciones respecto al desarrollo.

En el desarrollo del proyecto, se ha utilizado como interfaz de usuario una aplicación móvil desarrollada en Flutter desde cero, vinculándola a la base de datos Real-Time Database. Dado que Flutter es un SDK multiplataforma, al realizar el proyecto mediante este lenguaje, en caso de que los dispositivos para mostrar la interfaz de usuario sean de distinta plataforma, no será necesario desarrollar nuevas aplicaciones para cada tipo de plataforma.

En cuanto al entorno, de programación, habrá que valorar qué sistema operativo se posee y qué tipo de dispositivos se desea utilizar para ejecutar la aplicación.

- En el caso de querer ejecutar la aplicación en dispositivos con sistema operativo Android, tenemos el lenguaje propio como Kotlin, Java y lenguajes de programación web como HTML, CSS y JavaScript. La ventaja que poseen estos tipos de lenguajes es que pueden ser desarrollados en distintos entornos de programación, incluyéndose en este grupo los entornos de desarrollo multilenguaje más populares como Visual Studio Code o Eclipse, aunque el IDE más popular para desarrollar aplicaciones Android es en Android Studio. Otra ventaja que poseen estos entornos y lenguajes es que la mayoría de dispositivos funcionan en Android o son compatibles con el compilador, además de que, si no se posee un dispositivo móvil con el que ejecutar el código, podemos generar instancias de dispositivos virtuales de varias formas, siendo la más popular y típica es generar una instancia de dispositivo mediante Android Studio.
- Si la idea es generar la aplicación pensada para dispositivos de Apple iOS, desde hace años el lenguaje para el desarrollo de aplicaciones destinadas a iOS es Objective-C, pero desde hace unos pocos años, se lanzó Swift y está ganando mucha fuerza gracias al mayor rendimiento que ofrecen las aplicaciones generadas mediante este lenguaje, además de ser un lenguaje más sencillo que Objective-C. En cuanto a entorno de programación, existen muchos programas que contengan estos lenguajes, la problemática de iOS es que requiere de XCode para compilar el código, por lo que para cargar el código, será necesario poseer un dispositivo compatible con macOS o simularlo mediante una máquina virtual con VirtualBox, lo cual consume muchos recursos de la CPU, hasta el punto en que, si la máquina no es lo suficientemente potente y no cumple los requisitos mínimos, el acceso a la máquina virtual quede bloqueado por falta de recursos.

En cuanto a alternativas respecto al diseño y desarrollo de la interfaz de usuario, cabe mencionar la infinidad de posibilidades que ofrece actualmente la programación. La decisión tomada fue realizar una aplicación simple y entendible, que se pudiera utilizar como interfaz de usuario para la plataforma, donde se pudiera filtrar y visionar la información obtenida de los nodos. A partir de este punto, se han añadido algunas funcionalidades y paquetes como poder generar gráficos, aplicar filtros en los gráficos y pantallas de monitorización. Se pensaron añadir muchas funciones que ampliaran el contenido de la aplicación, pero debido a que el contenido a añadir podía escalar infinitamente, se tomó la decisión de dejarla hasta el punto en el que se encuentra en la documentación.

## Anexo C. Explicación del montaje del nodo físico

El microcontrolador ATmega328 es un microcontrolador desarrollado por Atmel y perteneciente a la serie megaAVR de alto rendimiento, basado en arquitectura RISC.

• Como sensores, tomaremos el sensor de temperatura y humedad DHT11 y el sensor de calidad de aire MQ-135.

El sensor DHT11 es un sensor analógico de salida digital alimentado a 3.5 – 5 V que realiza lecturas de temperatura y humedad mediante el envío de tramas de datos de 40 bits.

El DHT11 con PCB contiene 3 patas, una destinada a la alimentación, una dedicada a la transmisión de datos y la toma de tierra.

Los 8 primeros bits corresponden a la información relacionada con la parte entera de la humedad, los siguientes 8 a su parte decimal, los siguientes 8 corresponden a la parte entera de la temperatura, los 8 siguientes a la parte decimal y los últimos 8 son los bits de paridad. Los bits de paridad se utilizan para garantizar el correcto funcionamiento del sensor. Si la medida ha sido tomada correctamente, el valor de los bits de paridad deben suponer iguales a la suma de el resto de bits.

Dado que la trama de datos se produce en función del tiempo, lo recomendable para este sensor es dejar un cierto tiempo entre mediciones, por lo que el fabricante recomienda esperar entre unos 4-5 segundos entre lecturas.

Dado que el uso de este tipo de sensor DHT11/DHT22 está muy extendido en aplicaciones tanto de IoT como proyectos Arduino, Adafruit nos brinda una librería DHT.h como extensión de su librería Adafruit Unified Sensor Driver Library, la cual nos facilita funciones para obtener las lecturas en las unidades deseadas.

El sensor MQ-135 es el sensor de respuesta rápida y de alta sensibilidad para el control de calidad de aire. Su tensión de trabajo es 5V y la medición característica dependerá de la tensión que de en su salida de datos.

El MQ-135 posee 4 patas: una destinada a la alimentación, una toma de tierra y dos salidas de datos, una digital y otra analógica.

Orientado sobretodo en su uso en edificios y oficinas, para detectar algún tipo de fugas o alarmas de incendios. Dependiendo del tipo de sensor que tengamos, este puede poseer un potenciómetro para calibrar la sensibilidad del sensor de manera manual.

Para este tipo de sensor, a diferencia del DHT11, el valor de la lectura lo obtendremos directamente consultando el valor que se está transmitiendo a través de su entrada analógica, por lo que no requerirá tiempos de espera ni esperar a su inicialización.

• Para establecer la comunicación con el servidor, se ha elegido el módulo ESP8266 ESP-01 de Espressif Systems. Este módulo ofrece una solución Wi-Fi integrada muy económica, a día de hoy es la solución IoT más extendida en cuanto a desarrollos IoT mediante Arduino (NodeMCU lo integra ya de serie) y de bajo consumo.

Es necesario destacar que este módulo opera alrededor de los 2.5 – 3.6 V, por lo que habrá que tener mucho cuidado a la hora de realizar las conexiones de no alimentarlo con la misma alimentación que los sensores, que trabajan óptimamente a 5V.

Cabe mencionar, que por lo general, cuando utilicemos el módulo por primera vez, si no ha sido manipulado anteriormente, suelen traer una velocidad de transmisión de serie de 115200 baudios, por lo que o bien, podemos establecer la velocidad de transmisión de nuestro sistema IoT hasta la puerta de enlace a Internet a los 115200 los cuales nos viene de serie o, mediante comandos AT. El comando AT para modificar la velocidad, el comando es: "*AT+UART_DEF = baudios,8,1,0,0*" donde *baudios* es la velocidad a la que queremos transmitir. Por convenio, el Arduino suele trabajar a una velocidad de 9600 baudios, pese a que

También será necesario consultar la versión del SDK del firmware y asegurarnos de que la versión del SDK sea igual o superior a la versión 1.0.0, ya que es la versión mínima requerida para establecer la conexión con servidores regidos por determinados protocolos IoT. Si el módulo es adquirido recientemente, es muy probable que ya incluya una versión superior a la necesaria (ya que, a fecha de la redacción de este documento, la última versión del ESP-SDK es la 3.0). Para realizar la consulta del SDK, deberemos conectar el ESP8266 al programador (podemos utilizar la placa Arduino sin microcontrolador como programador) en modo de arranque (GPIO1 y GPIO2 desconectados) y, mediante una comunicación serial consultar la versión mediante el comando AT+GMR. El módulo nos devolverá la versión actual del Firmware y del SDK.

En caso de requerir modificar el SDK, deberemos descargar la última versión del SDK Firmware para el ESP8266 en la página web de Espressif (o cualquiera igual a superior a la 1.0.0 ya nos serviría para esta aplicación) y conectar el módulo ESP8266 en modo Flash (modo Flash: GPIO1 y GPIO2 conectados a tierra y alimentación) y procedemos a flashear la memoria del ESP8266.

## Anexo D. Software nodo físico. Ampliación técnica

• Finalmente, el código que deberá ejecutar el microcontrolador ATmega328P. Para que funcione el sistema como deseamos, el proceso estará pensado para inicializar los sensores para que recolecten datos del entorno y cargar las variables necesaria para la conexión, establecer una conexión a la red Wi-Fi mediante el módulo ESP8266 para que pueda enviar datos al servidor mediante mensajería a partir del protocolo MQTT (Message Queing Telemetry Transport) estableciendo un patrón de publicación y suscripción, realizaremos las lecturas de los sensores y las guardaremos en un tópico (en mi caso, lo he llamado "datos", representados mediante una cadena de carácteres)

Las librerías que utilizaremos para elaborar el código son:

- DHT: La utilizaremos para facilitar la toma de medidas mediante el sensor de temperatura y humedad DHT11, además de incluir las rutinas necesarias para transferir la trama de datos.
- Avr/io: Dado que nuestro controlador es AVR, esta librería nos facilitará el acceso a las definiciones y registros del microcontrolador.
- WifiEsp: Nos permitirá preparar la placa de Arduino para conectarse a Internet.
- WifiEspClient: Nos permitirá establecer un objeto como cliente.
- PubSubClient: Esta librería nos dará la posibilidad de tener un cliente para realizar mensajes de publicación y suscripción hacia un servidor MQTT.
- SoftwareSerial: La utilizaremos para asegurar una comunicación serial entre el módulo ESP8266 y el ATmega328P donde tengamos conectado el modulo.

El código constará de las siguientes partes:

- Declaraciones y definiciones globales:
- Importamos las librerías comentadas anteriormente.
- Definimos los parámetros de la red WiFi (nombre de la red y contraseña)
- Declaramos y definimos en qué pines se transmitirán los datos de los sensores (DHT11 y MQ-135). Para el caso del sensor DHT11 también deberemos declarar qué tipo de sensor tenemos, ya que la librería de Adafruit incluye otros modelos de sensores dedicados a este fin y para acceder a estos métodos debemos declarar un objeto de la clase DHT indicando el pin digital en el que tenemos los datos conectados y el tipo de sensor instanciando el DHTTYPE.
- Estableceremos la IP o nombre en el que se alojará nuestro servidor MQTT.
- Generaremos un objeto como cliente.
- Estableceremos una comunicación serial entre controladores.
- Declaramos un contador para el envío de datos y tener localizada la última lectura.
- Una variable de estado, para en caso de que haya algún error, nos sea comunicado.

- Función de setup (inicialización):

- Iniciar la comunicación serial a los mismos baudios que hayamos configurado el módulo ESP8266 para tener un log donde saber en cada bucle qué acción se está realizando y qué respuesta está obteniendo.
- Iniciamos la lectura del sensor DHT conectado mediante su método de inicio.
- Iniciamos la conexión a la red WiFi.
- El objeto cliente que hemos declarado para que se le pueda alojar el servidor MQTT, utilizaremos su método setter para que se enlace a la IP del servidor y el puerto 1883 (más adelante aclararé el uso de este puerto).

- En el bucle tendremos:

- Una primera validación para ver si el módulo ESP8266 está conectado a la red WiFi.
- En caso de que la validación no se cumple, debe llamar a una función para reconectar a la red WiFi.
- A continuación, realizaremos una validación para ver si el cliente está conectado con el servidor. En caso de que no esté conectado, se llamará a una función para reconectar el cliente al servidor.
- Si se han cumplido ambas validaciones, se podrá ejecutar la función para enviar datos. Cabe recordar, que en esta parte, dado que el sensor DHT11 necesita unos segundos de delay entre tramas.
- Una vez enviados los datos, podemos reiniciar el cliente, finalizar el mensaje e iniciar el siguiente mensaje.

- Las funciones mencionadas consisten en:
- Función para inicializar la red WiFi: Iniciamos la comunicación serial con el software para recibir una respuesta en caso de que tengamos un dispositivo serial. Iniciamos la conexión WiFi del software. En caso de que no se haya podido realizar la conexión WiFi, se mostrará el tipo de error mediante el monitor serial, en caso de que no se haya conectado satisfactoriamente. Se haya podido realizar la conexión o no (a excepción de no tener o no detectar ningún módulo WiFi conectado, en ese caso, la función permanece en espera hasta que lo detecte), iremos a la función de reconectar WiFi.

- Función para reconectar a la red WiFi: Una vez hayamos inicializado la conexión a la red WiFi, conectaremos el módulo ESP8266 a nuestra red WiFi doméstica mediante las credenciales que hemos inicializado en las variables globales. En caso de que las credenciales no sean correctas o haya un fallo de conexión en esta parte, reconectaremos el módulo mediante repeticiones en esta función. Una vez se haya producido la conexión, salimos de la función y continuamos con la ejecución en el loop.
- Función para reconectar el cliente: Si no se detecta que el cliente se haya conectado, la función se repetirá en bucle hasta que el cliente conecte notificando del tipo de error que encuentra al conectar con el servidor. Para realizar la conexión, generamos el ID del cliente (podemos poner cualquier nombre) y, en caso de requerir autentificación para acceder al servidor, en el método de conectar añadimos las credenciales de nuestro usuario para acceder de forma segura, junto con el ID del cliente. Si la conexión ha sido exitosa, salimos de la función en cambio, si la conexión ha sido fallida, se nos notificará el tipo de error y seguiremos en el bucle de la función, hasta lograr conectar con el servidor.
- Función para publicar los datos al servidor: En este caso, primero se realizarán las medidas correspondientes a los sensores DHT y MQ-135. Dicha información tomada de las medidas, se guardará en una cadena de caracteres en el formato que se desee. En este caso, lo guardaré en una notación parecida a JSON, sin encapsular para facilitar la creación del objeto referente a las medidas. Una vez generada la cadena de caracteres con la información que queremos enviar, publicaremos la información mediante el objeto cliente que hemos generado para que se pudiera alojar al servidor MQTT. Cabe mencionar, que este mensaje que contiene la información que estamos enviando (llamado "*topic*"), debe recibir un nombre de identificación con el que el bróker identificará al mensaje (llamado "*payload*"), en este caso, el nombre del topic que recibirá el bróker es "datos". Una vez publicada la información, salimos de la función y sigue la ejecución del bucle.

## Anexo E. Conectividad. Servidor MQTT. Ampliación técnica

Como ya se ha mencionado anteriormente, el protocolo de comunicación que se ha utilizado para comunicar el nodo físico con un servidor es mediante el protocolo MQTT. Es importante mencionar que el protocolo MQTT, al ser un protocolo de mensajería, está basado en la pila TCP/IP (Protocolo de Control de Transmisión en redes), de manera que las conexiones que realiza entre clientes se mantiene abierta y, en caso de realizar una nueva petición con el mismo cliente, volverá a utilizar la misma conexión que se ha mantenido abierta, en lugar de abrir una nueva. La ventaja de este tipo de conexiones TCP es que, una vez se ha realizado la conexión entre servicios, es que los datos pueden ser enviados de manera bidireccional. Esto último es solo un apunte, en este proyecto, la información irá en un único sentido.

De esta manera, gracias al módulo ESP8266 E-01, una vez realizada la conexión con el servidor donde está alojado el MQTT como oyente, el cliente del ESP8266 E-01 (nodo) quedará suscrito al bróker y podremos suscribir otro cliente/servicio al bróker para obtener la información (*payload*) que el nodo físico está publicando con el identificador (*topic*) de "datos".

La forma en la que hemos añadido este bróker MQTT de datos ha sido mediante el uso de Eclipse Mosquitto, un bróker MQTT de código abierto que nos permite acceder a los métodos de publicación y suscripción característicos de este modelo:

- En primera instancia, deberemos descargar e instalar Eclipse Mosquitto en nuestro ordenador (ya sea ordenador local, Raspberry Pi o cualquier placa computadora que contenga un sistema operativo que pueda ser compatible con el software de Mosquitto).
- Una vez instalado, lo podremos alojar como oyente en uno de los servidores disponibles. La lista con todos los servidores disponibles se podrá encontrar en la bibliografía, en este caso se utilizará el puerto 1883, que incluye el protocolo MQTT, sin encriptación de datos y no requiere autentificación. En caso de que se desee encriptar los datos sobre TLS para conservar una mayor seguridad en los datos, se deberá configurar en el grupo de puertos 8883. En caso de querer utilizarlo como bróker intermediario en WebSockets, se podría poner en escucha en el puerto 8080 (dependerá del tipo de encriptación y autentificación que queramos).
- Si la versión de Mosquitto es superior a la 2.0.0, esta versión no permite usuarios anónimos, por lo que, en caso de que no queramos autentificación, habrá que ir al fichero de autentificación, abrirlo como administrador y añadir un "*listener*" a nuestra IP de red (o hacia cualquiera, declarándola como 0.0.0.0) y añadir una línea de confirmación en el fichero que permita anónimos como verdadero.
- Hecho este paso intermedio, se utilizará la herramienta Docker para automatizar el despliegue del servidor MQTT, utilizando como imagen la última versión de Eclipse-Mosquitto, de forma que mediante el Docker-Desktop se podrá alojar y desalojar el servidor dándole a un botón y se actualizará a la última versión de Mosquitto automáticamente (lo cual aveces, puede resultar un problema, ya que dependiendo de las versiones, puede que modifiquen los permisos de acceso al bróker y en cualquier momento deje de funcionar y se deba optar a modificar el archivo config donde hayamos instalado Mosquitto o utilizar una imagen de Docker-Compose de una versión anterior y mantener la versión estática). Cabe mencionar que los puertos a los que publicaremos los puertos para el servidor MQTT serán el 1883 y el 9001, este último lo utilizaremos, ya que es el puerto que contiene el protocolo TCP/IP.

## Software. Nodo remoto alternativo: AEMET

Para la generación del primer nodo remoto, se ha decidido consultar la página de la Agencia Estatal de Meteorología pidiendo el acceso como desarrollador mediante una API Key. La API que se decidió consultar (Barcelona), al obtener la IP en la que está registrada, presenta el contenido en formato XML y contiene objetos con atributos tales como "temperatura" y "humedad_relativa", además de añadir una fecha y hora a la que se destinaba la predicción. El contenido de la API se actualiza cada 24 horas y presenta los datos del mismo día en el que se realiza la consulta hasta seis días más.

El objetivo para este nodo remoto es lograr generar un objeto a partir de los campos seleccionados en el archivo que se muestra en la API "temperatura", "humedad_relativa" y "fecha" para el día en el que se realizara la solicitud.

Para lograrlo, se ha utilizado Node-RED, que nos permite utilizar métodos HTTPS, generar automáticamente bloques de JavaScript, incluir funciones y utilizar funciones:

- En primera instancia, debemos obtener el contenido de la API, por lo que dando como URI la dirección http en la que se encuentra la API, realizamos una petición HTTP GET para obtener como retorno el contenido en formato UTF-8 String.
- Una vez realizado el primer paso, obtendremos una cadena de caracteres de más de 12000 caracteres. Para hacer de esta cadena de caracteres, algo manejable, realizaremos un parseo de String en notación XML a objeto.
- Cuando tengamos el objeto, podremos empezar a filtrar por sus atributos. Una vez lleguemos a la información de la predicción del día, separaremos el objeto en tres objetos distintos, un objeto que contenga en su interior los atributos de "temperatura", otro que contendrá en su interior los atributos de "humedad_relativa" y por último, guardaremos la fecha.
- Dado que el formato en el que se presentan los datos incluye caracteres que dan problemas en ciertos lenguajes como "$" y "_", cambiaremos de formato objeto a formato String en notación JSON y mediante una función buscaremos si el String contiene los caracteres problemáticos y los sustituiremos por cadenas de caracteres o carácteres identificables.
- Una vez conseguido el formato que queremos, generamos un objeto JSON de cada parámetro que hemos buscado y los unimos para formar un solo objeto en formato JSON.

De esta forma, se obtiene mediante la API de AEMET, un objeto que contiene los datos que nos interesaba filtrar y en formato JSON.

## Anexo F. Ampliación técnica. Bases de datos

### Firebase Real-Time Database

El almacenamiento y sincronización de los datos es realizado mediante la Real-Time Database de Firebase. Se trata de una base de datos NoSQL alojada en la nube, cuya notación requerida para el almacenamiento y gestión de los datos es formato JSON. La sincronización de los datos, como su nombre indica, es en tiempo real para todos los clientes conectados a la base de datos, además de permitir el acceso a los datos sin conexión a internet, permitiendo la sincronización automática con los datos más recientes al recuperar la conexión. El acceso a los datos para realizar operaciones CRUD (Crear, Leer, Actualizar y Borrar), permite realizarse o bien mediante la API de Real-Time Database o desde aplicaciones cliente. Posee compatibilidad para lenguajes de desarrollo de aplicaciones multiplataforma y web.

Esta base de datos es considerada en desarrollos de Internet of Things, para determinadas arquitecturas poseedoras de software y librerías de autentificación que permiten el envío de los datos recolectados a nivel de dispositivo directamente hacia la API. La ventaja que nos ofrece este tipo de procedimiento es que, gracias a las funcionalidades que posee la base de datos al estar sin conexión, nos permite sincronizar los datos recolectados sin necesidad de tener servidores intermedios. En este proyecto, dada la arquitectura del dispositivo elegida, para el envío hacia la base de datos se requiere de servidores intermedios y un protocolo de telecomunicación.

Los motivos por los cuales se ha elegido esta base de datos como almacenamiento de datos son los siguientes:

- Dado que el proyecto consta de una red de nodos repartidos en una zona geográfica que puede ir creciendo y, por lo tanto, se podrán añadir mayor número de nodos, al elegir una base de datos de tipo NoSQL, podremos hacer que el proyecto sea escalable en cuanto a número de dispositivos conectados, tanto por parte de los nodos, como por parte de los clientes (usuarios) conectados.
- Es una base de datos que actúa como servicio en la nube, que además, se puede utilizar como una API mediante comandos HTTP.
- Aporta una notación orientada a objeto (JSON) para la transmisión de datos, que es actualmente la más extendida, flexible y fácil de usar.
- La finalidad de las aplicaciones del cliente es realizar lecturas simples hacia la base de datos, sin necesidad de ordenamientos avanzados (se utilizará el mismo orden de llegada y, como mucho, se aplicarán filtros y selecciones que extraerán los datos en orden para ser procesados mediante algún tipo de función dentro de la propia aplicación del cliente.
- Se requiere que la base de datos posea una disponibilidad alta, pero no necesariamente crítica, es suficiente con que los datos se actualicen y sincronicen en tiempo real, con baja latencia.
- Para la realización del proyecto, únicamente se necesita de una sola base de datos, por lo que se utilizará solamente una instancia.
- El desarrollo de los datos vendrán con el mínimo de capas posibles, por lo que resultará cómodo el uso de instancias para filtrar los datos almacenados en la base de datos y actualizarlos en tiempo real en la aplicación.
- Real-Time Database no posee límites de escritura o lectura diarios de forma gratuita, a diferencia de Cloud Firestore de Firebase, que admite hasta un límite de

10.000 operaciones diarias de forma gratuita, lo cual puede ser un problema a la hora de gestionar grandes cantidades de datos.

- La base de datos resulta fácilmente vinculable a dispositivos que permitan ejecutar aplicaciones móviles o web mediante distintos tipos de lenguaje multiplataforma.
- Al vincular la aplicación al entorno de Firebase, tendremos acceso a los paquetes que ofrecen los lenguajes de programación multiplataforma para el propio entorno de Firebase.
- Uno de los paquetes más extendidos para este tipo de aplicaciones (Firebase Auth) permite dar mayor rendimiento a una de las funcionalidades que tiene el entorno de Firebase, que es el de autentificación de usuarios. En caso de que en un futuro se desee requerir autentificación, dar servicios a ciertos usuarios o gestionar quién puede acceder a la base de datos, combinar la autentificación que ofrece la consola del proyecto en Firebase junto al paquete correspondiente permite aplicar dichos servicios de forma simple y eficiente.
- Por último, cabe mencionar que Firebase, desde hace unos años, pasó a ser dominio de Google, por lo que, al desarrollar una aplicación en Flutter (Dart), siendo también un lenguaje desarrollado por la propia Google, ofrece mucha compatibilidad en cuanto a servicios se refiere.

**Aclaración escrita del esquema de la base de datos (*Figura 25).***

- Referencia de la base de datos: representa la instancia creada para referenciar la base de datos.
- Rutas (Nodos): permiten separar en diferentes rutas de acceso hacia los documentos.
- Documentos: representan el conjunto de datos en notación JSON escritos en la base de datos distinguidos para cada uno de los nodos
- Datos: incluyen las mediciones realizadas por los nodos, junto con las fechas agregadas al acceder a los servidores. En otras palabras, representan los atributos de los objetos que forman cada uno de los documentos.

Las operaciones de escritura de la base de datos se realizan mediante el servidor de Node-RED, utilizando nodos http request con el método POST para que envíe los datos en notación JSON hacia el enlace donde está alojada la API de la ruta del nodo al que corresponden dichos datos.

Las operaciones de lectura son realizadas mediante la aplicación que actúa como interfaz de usuario, el procedimiento de lectura en la aplicación consiste en, tras haber vinculado el proyecto de la aplicación a la consola de la base de datos de Firebase, crear una instancia, en las pantallas las cuales la vayamos a utilizar, que sirva de referencia hacia la base de datos, escoger una de las rutas (nodo) de donde obtener los documentos y, en caso de que no queramos obtener todo el conjunto de documentos, proponer un límite de cuales filtrar. Este procedimiento generará una query a partir de la cual podremos obtener los datos de dicha ruta en formato JSON.

# Anexo G. Ampliación interfaz de usuario

## UI. Funcionalidades

### 1. Problema

Se requiere obtener una interfaz de usuario interactiva para poder visualizar los datos obtenidos a partir de la información dada por distintos nodos, almacenada en una base de datos en Firebase.

### 2. Solución

Crear una aplicación que permita la visualización de los datos almacenados, realizar un filtrado en función de la localización de los nodos y en función de sus fechas.

### 3. Funcionamiento general

Acceso al servicio en la nube:

La aplicación estará conectada a un servicio en la nube (Firebase) para vincular a los usuarios de la aplicación con la base de datos. Al arrancar la aplicación, se establecerá dicha conexión al servicio y obtendremos una referencia de acceso a la base de datos a partir de la cual extraeremos la información.

Navegación entre pantallas:

El modelo de enlace y navegación entre pantallas será mediante rutas nombradas.

Acceso a la información de los nodos:

- Mediante el muestreo de una lista con el nombre de todos los nodos a los que se puede acceder pulsando sobre el nombre del nodo.
- A través de un mapa donde se mostrarán los puntos geográficos donde están situados los nodos recolectores de información. Al pulsar sobre los nodos, se accederá a la información del nodo.

Información de los nodos:

Cuando accedemos a la información de un nodo, nos dirigiremos a la pantalla del nodo, donde podremos ver la fecha y última medición que se ha realizado en dicho nodo. La información de la última medición se actualizará en tiempo real.

En esta misma pantalla, se podrá acceder a distintas pantallas que poseerán una lista de todas las mediciones registradas en la base de datos.

Desde esta última pantalla también podremos acceder a una pantalla donde visualizar gráficos que representan la evolución temporal de las mediciones, los cuales se podrán filtrar en función de la fecha seleccionada.

### 4. Función del usuario

El proyecto no requiere de ningún servicio específico para el usuario, ya que la función principal de la aplicación consiste en proveer al usuario de una interfaz gráfica al usuario.

El acceso a la lectura y escritura de la base de datos se realiza de manera pública, de forma que tampoco será necesario un factor de autentificación previo para acceder a la base de datos.

Por lo tanto, la función del usuario consistirá en la capacidad de navegar entre pantallas, elegir los datos que se desean mostrar y elegir los filtros a aplicar en los datos mostrados, dando acceso de forma anónima y sin guardar su registro.

En caso de querer dar acceso a servicios de usuario y registro en el proyecto, quedará anexado una pequeña ampliación del diseño sobre como añadir este tipo de servicios de autentificación, registro y actuación sobre la identificación de usuario conectada, mediante el paquete de autentificación de Firebase.

## 5. Control de errores

Dado que la aplicación consiste en realizar la función de interfaz de usuario para un muestreo de datos. Dentro de la aplicación tendremos que tener en cuenta los siguientes posibles errores:

- Realizar un correcto filtrado de datos de la base de datos, de manera que sea legible y corresponda a los filtros correctos.
- Asegurar que los datos filtrados en los gráficos no produzcan errores debido a duplicados.
- Comprobar que las rutas de navegación de usuario entre pantallas sean las correctas.
- (Para nodos virtuales) Establecer un control de datos objetos duplicados en la base de datos.

## UI. Modelo de datos

Formato en base de datos:

**• Nodo Físico:**

- Fecha. El tipo de dato es una cadena de caracteres en formato ISOString. Representa la fecha y hora en la que el servidor de Node-Red ha recibido el mensaje del nodo físico a través del servidor MQTT tras haber realizado la medición.
- Temperatura. Dato de tipo real, admitiendo decimales. Representa el valor numérico de la medición de temperatura tomada mediante el sensor DHT11 del nodo físico.
- Humedad. Dato de tipo entero. Representa el valor numérico de la medición de humedad tomada mediante el sensor DHT11 del nodo físico. El dato se guarda en la base de datos como entero, pero posteriormente se utilizará como real en la aplicación. Esto es debido a que las mediciones de los nodos virtuales poseen mayor precisión en la medición, ofreciendo un grado de magnitud mayor.
- PPM: Dato de tipo entero. Representa el valor numérico de la medición de temperatura tomada mediante el sensor MQ-135 del nodo físico.

**• Nodo Remoto:**

- Fecha. El tipo de dato es una cadena de caracteres en formato ISOString. Representa la fecha y hora de la última medición que nos ofrece la web API de la cual obtenemos los datos.
- Temperatura. Dato de tipo real, admitiendo decimales. Representa el valor numérico de la medición de temperatura tomada de la localización geográfica obtenida de la web API.

- Humedad. Dato de tipo real, admitiendo decimales. Representa el valor numérico de la medición de humedad tomada de la localización geográfica obtenida de la web API.

**Formato en la aplicación:**

Siguiendo la documentación del paquete añadido de gráficos, se recomienda crear una nueva clase con dos atributos. Para simplificar esta clase y hacerla polivalente para cualquier tipo de medición.

**• Datos gráfico:**

- Dato. Dato de tipo real, admitiendo decimales. El valor de este atributo representará el valor dado en el eje Y del gráfico en un punto determinado. Representa cualquier tipo de dato que queramos graficar en el eje vertical.
- Tiempo. Cadena de datos de caracteres en formato ISOString. El valor de este atributo representa el punto respectivo en el eje X. El atributo nos servirá para dar valores de tiempo de forma periódica a los datos del eje X.

**UI. Planificación del desarrollo de la aplicación**

- Establecer modelo de datos necesario
  En primera instancia, habrá que decidir el modelo de datos sobre las funciones que deberá desempeñar la aplicación.

- Desarrollo previo para la obtención de datos
  Este apartado ocupa el contenido desde el desarrollo de los nodos físicos hasta los servidores MQTT y Node-Red utilizados para el envío a la base de datos.

- Recopilación de mockups e ideas para la interfaz de usuario
  Consiste en recopilar ideas y referencias para realizar los diseños de las pantallas, anotar ideas para su navegación y empezar a pensar qué funciones desarrollar para cada tipo de pantalla.

- Prototipos de pantallas en Flutter para las pantallas más relevantes
  Desarrollo de pantallas añadiendo las propuestas de los mockups e ideas para generar la interfaz en Dart para las primeras pantallas donde se podrán realizar posteriormente las primeras pruebas de funcionamiento (pantalla de nodo físico).

- Establecer rutas de navegación entre pantallas y interactuaciones entre ellas
  Siguiendo el diagrama de navegación entre pantallas, se implementarán las rutas en la aplicación.

- Preparar las pantallas para gestionar datos siguiendo el modelo de datos
  Una vez se haya obtenido la visualización y el sistema de navegación entre pantallas, para dar utilidad al modelo de datos elegido, se generan funciones que, como retorno, devuelven el dato que sea necesario.

- Pruebas de funcionamiento mediante listas de datos locales
  Tras realizar el diseño de las pantallas, aplicar rutas de navegación y funciones para aplicar el modelo de datos, se harán comprobaciones de funcionamiento inicializando el modelo mediante variables como listas de datos de prueba.

- Agregar proyecto de la aplicación a la Real-Time Database de Firebase

  Cuando se haya realizado la comprobación del correcto funcionamiento de la aplicación mediante datos de prueba, podremos vincular la aplicación a la base de datos, exportar los paquetes necesarios a nuestra aplicación, actualizar la versión mínima del SDK a una versión compatible, añadir las dependencias necesarias y añadir los servicios de Google a la aplicación (seguir los pasos recomendados al crear el proyecto en Firebase).
  En este apartado también se añadirán las funciones de búsqueda de datos a partir de nuestra referencia hacia la base de datos.

- Realizar pruebas de funcionamiento mediante el modelo de datos establecido a partir de la base de datos

  En primera instancia, comprobar que la base de datos permite escribir y leer en la base de datos a través del proyecto de la aplicación para confirmar la correcta conexión entre el proyecto en Firebase y la aplicación.
  A continuación, eliminar las listas de datos locales y modificar las funciones para utilizar los datos obtenidos a partir de la base de datos.
  Realizar una segunda comprobación para ver que las listas se obtienen en correctamente y en el orden correcto.

- Implementar pantallas de gráficos

  Una vez hecha la comprobación de que el cuerpo de la aplicación funciona correctamente y se conecta correctamente a la base de datos, se empezará a desarrollar e implementar el resto de pantallas que también utilicen los datos aplicados a una función (filtros).

- Implementar pantallas de monitorización

  Tras terminar la pantalla de gráficos, a continuación, se crearán las pantallas más simples, que servirán para monitorizar la última medición de uno de los nodos (a elección del usuario).

- Depuración de errores

  Resolución de los principales errores que pueda contener la aplicación para la visualización de datos, aplicar filtros y selección de pantallas.

- Simplificar la aplicación para hacerla fácilmente y rápidamente replicable

  Facilitar al máximo la adición de nuevos nodos para que, mediante mínimas modificaciones, se puedan añadir y visualizar nuevos nodos que recolecten datos del mismo tipo que los ya contenidos de la aplicación.

- Añadir opciones extra o auxiliares

  Para finalizar, añadir mejoras para la calidad de uso y manejo para el usuario.

## UI. Diagrama de versiones

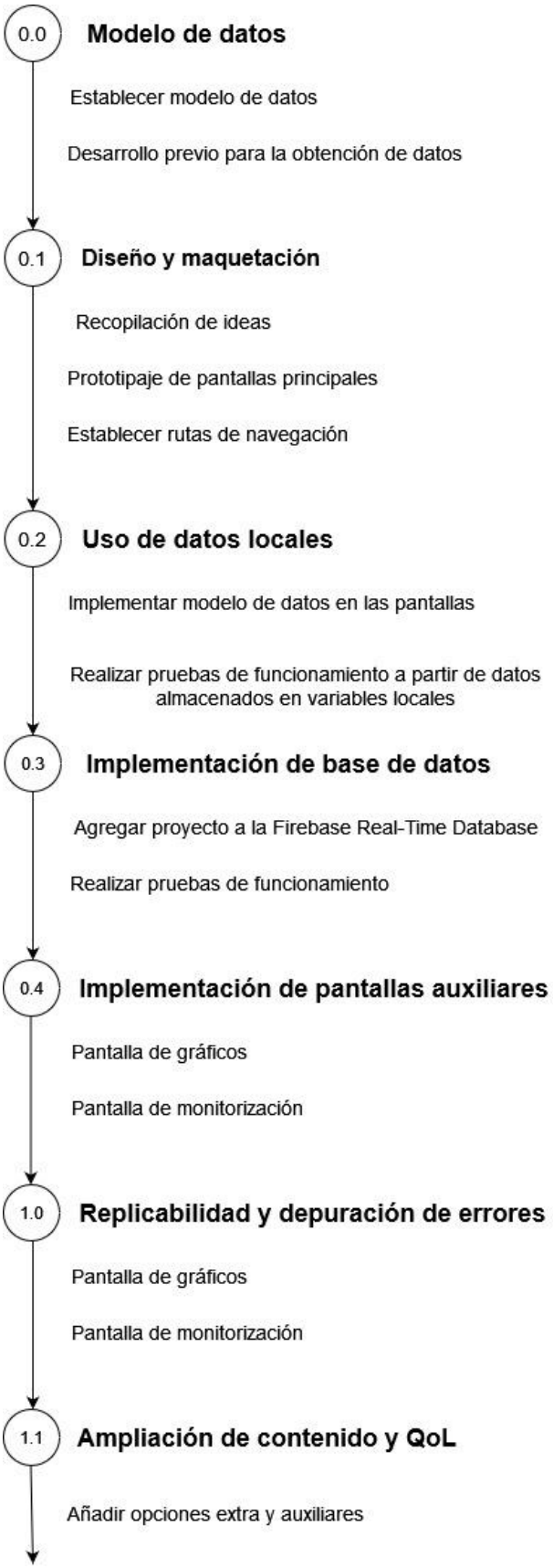


*Figura 42:* Rafael C. (2022). [Diagrama de versiones de la aplicación]

## UI. Diseño de pantallas (todas las que faltan en la memoria)


*Figura 43:* Rafael C. (2022). [Prototipo de Pantalla: Pantalla lista de humedad]

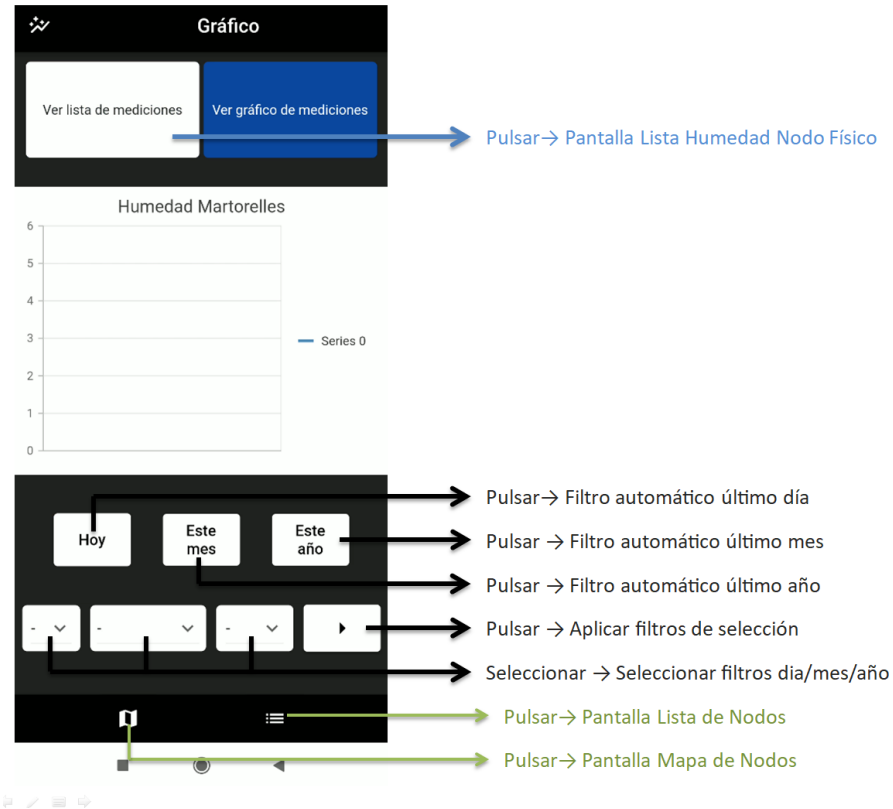


*Figura 44:* Rafael C. (2022). [Prototipo de Pantalla: Pantalla gráfico (humedad)]

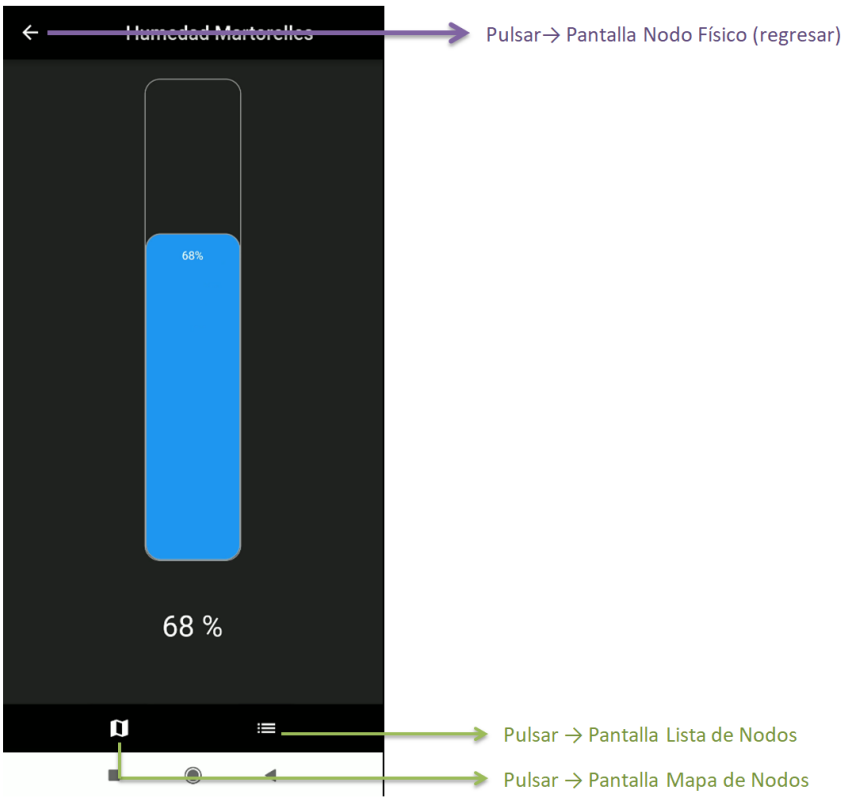


*Figura 45:* Rafael C. (2022). [Prototipo de Pantalla: Monitor de humedad]


*Figura 46:* Rafael C. (2022). [Prototipo de Pantalla: Pantalla Lista CDA]


*Figura 47:* Rafael C. (2022). [Prototipo de Pantalla: Pantalla todas las mediciones]

# Anexo H. Seguridad de dispositivos y sistemas

## Seguridad de dispositivos

Debido a que nuestro dispositivo IoT (nodo físico) se encuentra conectado a la red (Internet), puede ser expuesto a uso por terceros, lo cual puede concurrir en que los datos obtenidos sean extraídos por otros usuarios sin consentimiento previo.
En caso de querer proteger el acceso a los datos de los dispositivos para evitar estos casos, podemos actuar de distintas formas.
Siguiendo los consejos que nos ofrece la empresa de consultoría y servicios digitales avanzados de la UPC (IThink UPC) para segurizar dispositivos IoT:

- Cambiar las credenciales predeterminadas. Para cumplir con este punto, conociendo que de forma predeterminada, la librería PubSubClient, a no ser que se le especifique, conecta con el cliente automáticamente de forma anónima. Esto se resuelve aplicando una ID y una contraseña a la variable cliente del programa del microcontrolador que coincida con las credenciales que pide el servidor MQTT (por lo tanto, no debemos modificar el archivo de configuración para admitir uso anónimo, como se ha mencionado en el desarrollo del servidor Mosquitto, desde la versión 2.0 el bróker pide autentificación por defecto). Por lo tanto, también habrá que configurar el nodo MQTT in de Node-Red, modificando las credenciales de seguridad (pestaña Security del nodo).
- Actualizar firmware a su última versión. Para el caso del proyecto, el firmware a actualizar es el del módulo WiFi ESP8266 ESP-01, para realizarlo podemos utilizar un programador o la propia placa de desarrollo Arduino UNO/Genuino UNO sin microcontrolador, conectando el módulo en modo debug (GPIO0 a tierra y GPIO2 con señal) y cargar la última versión compatible del firmware (disponible en la web de Espressif) mediante un flasher.
- Instalar las actualizaciones de las aplicaciones de nuestros dispositivos IoT. Para este caso, habrá que prestar atención si las librerías han recibido alguna actualización por parte del desarrollador y, en caso de que sea necesario, aplicar las modificaciones necesarias en el código para el correcto funcionamiento del dispositivo.
- Segmentar la red para los dispositivos IoT. En este caso, al tratarse de un prototipo académico, se utiliza una red local, pero sí que es una buena práctica a realizar, en caso de tener otros dispositivos conectados a la misma red que los dispositivos IoT.
- Deshabilitar o proteger el acceso remoto al dispositivo IoT. En el estado actual del proyecto, no se ha programado ninguna función de acceso remoto. En caso de seguir escalando el proyecto para realizar este tipo de funciones, sí que habrá que tener en cuenta una función que deshabilite o proteja accesos que permitan el control a distancia.

**Seguridad de base de datos**

La seguridad que ofrece la base de datos para el caso de la Real-Time Database de Firebase o, en general, el entorno de Firebase, viene dada en gran parte por sus reglas.

Con las reglas podemos determinar quién puede leer o escribir en la base de datos, mediante código. Estas reglas se pueden complicar o simplificar a necesidad de cada tipo de aplicación.

En el desarrollo del proyecto, se han utilizado dos tipos de reglas:

- Reglas públicas, donde cualquier usuario fuera anónimo o no podía leer o escribir en la base de datos.

- Privatizar la escritura en la base de datos, para que todo el mundo pueda leer el contenido de la base de datos, pero únicamente los usuarios registrados puedan escribir en la base de datos.

De esta manera, al activar la autentificación que proporciona Firebase Auth, se ha registrado un solo usuario, a través del cual se puede escribir en la base de datos.

Existen muchas otras reglas y formatos de declararlas, ya sea mediante condicionantes, validaciones de datos, elementos secundarios indexados a los cuales acceder, uso de métodos, funciones…

Hay que tener en cuenta, que al modificar las reglas públicas a privatizarlas, se modifica el acceso a la escritura (o a la lectura, si también se ha privatizado) para usuarios no autentificados, por lo que para el desarrollo del flujo habrá que añadir funciones de autentificación en los nodos http request. Otra opción es instalar nodos extra que realicen esta función, mediante la opción de añadir nodos a la paleta, ya que existen diversos nodos instalables destinados a trabajar en conjunto con las bases de datos y web API de Firebase.

## Anexo I. UML. Descripción del software a tiempo real

A través de lenguaje unificado de modelado, se realizará una representación sobre el modelado del software para el desarrollo de este proyecto de manera conceptual.

Para representar el sistema desarrollado, se han realizado diversos diagramas estructurales. En los Anexos se han adjuntado los diagramas UML que no han sido utilizados para complementar explicaciones en la memoria:

• **Diagrama de secuencia**

Cabe mencionar que, para el caso del diseño para el diagrama de secuencia, tanto el software continúa su ejecución y se siguen disparando eventos.

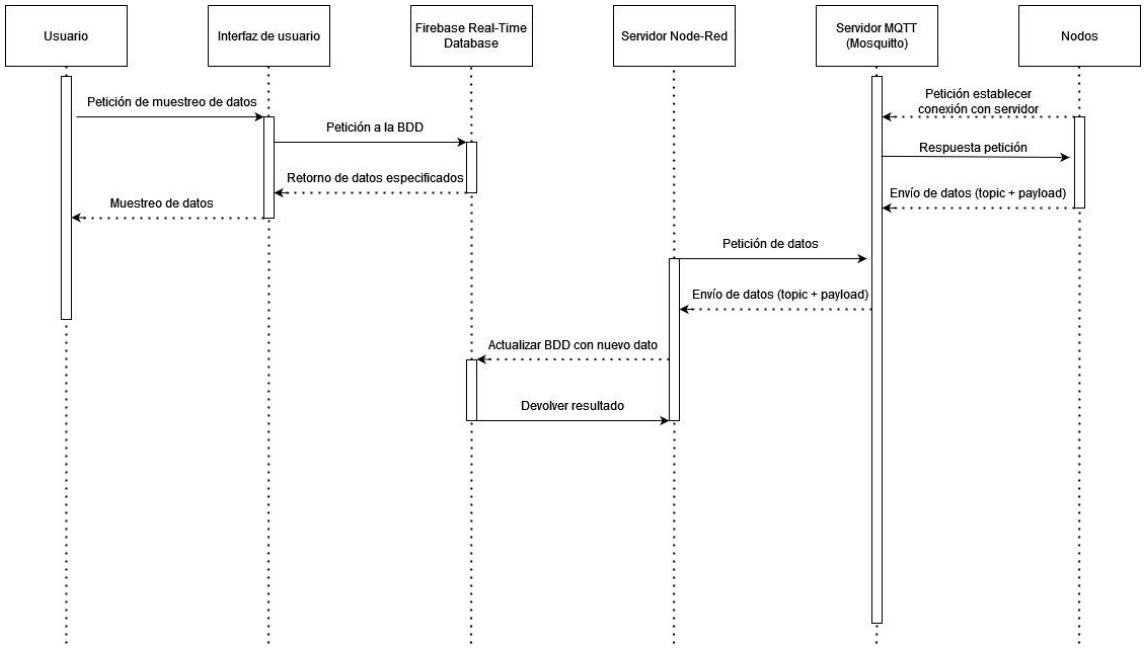


*Figura 48:* Rafael C. (2022). [Diagrama de secuencia]

• **Diagrama de estados**

El resto de diagramas han sido utilizados y referenciados en la memoria, faltaba por completar el diagrama de estados del nodo físico:

- Servidores: Nodo físico

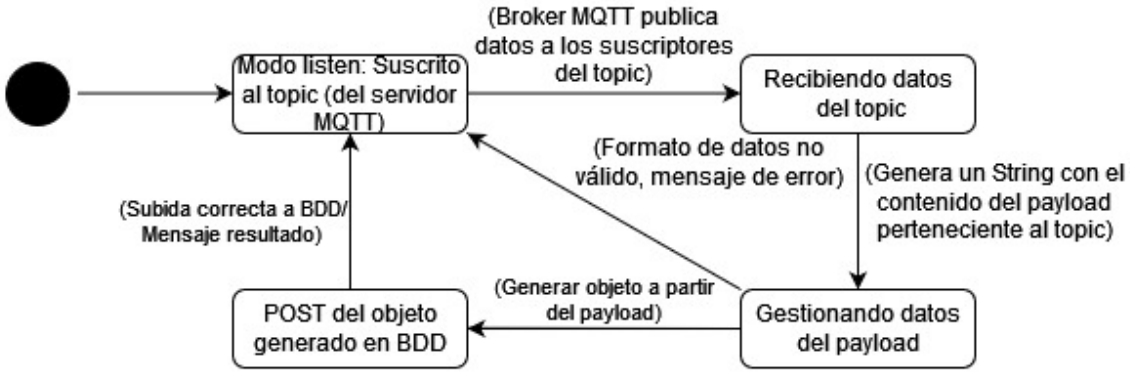


*Figura 49:* Rafael C. (2022). [Diagrama de estados: Servidores (Nodo físico)]

• **Diagrama de actividad**

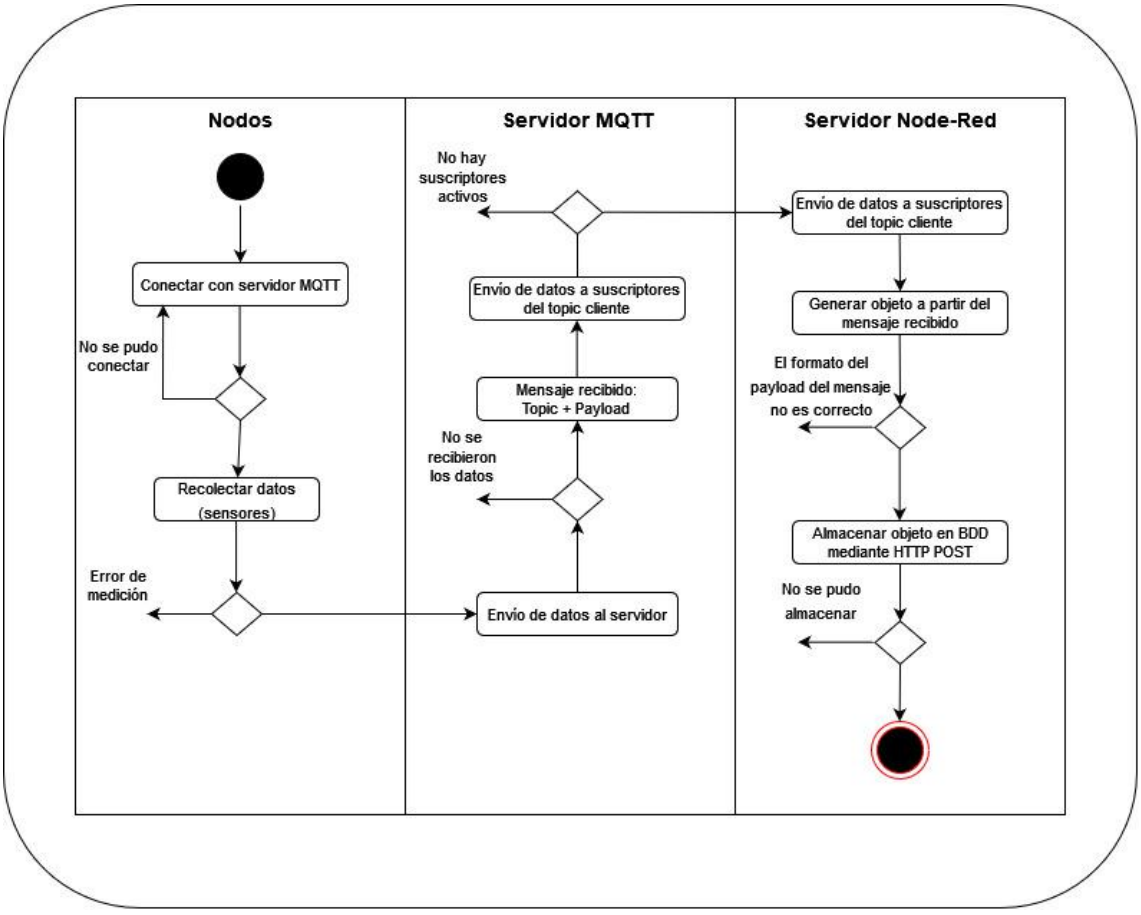


*Figura 50:* Rafael C. (2022). [Diagrama de actividad del sistema]

• **Documento descriptivo**

**Nombre: Sistema red de nodos**

**Autor:** Rafael Callejón Rodríguez

**Fecha:** 26/04/2022

**Descripción:**

Permite establecer una red de nodos recolectores de datos, cuyos datos podrán ser monitorizados por el usuario mediante una interfaz de usuario.

**Actores:**

- Usuario

**Precondiciones:**

- En este caso, el sistema no requiere de autentificación a los nodos para acceder a los servidores, ni la interfaz de usuario requiere de autentificación. En caso de querer privatizar el acceso en cualquiera de los puntos de acceso al sistema, será necesario apuntar la precondición.
- Los nodos estar en correcto funcionamiento y tener un punto de acceso (sea mediante cable o inalámbrico) a los servidores.
- El usuario debe tener acceso a la interfaz gráfica.

**Flujo normal:**

- El usuario realiza una consulta de los datos aplicando los filtros requeridos por el usuario mediante la interfaz de usuario.
- La interfaz del sistema muestra al usuario una pantalla en la cual aplica los filtros requeridos a la base de datos.
- De forma asíncrona, los nodos irán recolectando datos y los irán enviando como mensaje al servidor MQTT al cual están suscritos.
- A medida que el servidor MQTT va recibiendo los datos de los nodos, el mensaje con el payload que contiene los datos, se comprueba la validez de los datos y son procesados en el servidor de Node-Red.
- Una vez procesado el mensaje, se genera un objeto con los contenidos del mensaje que es enviado a la base de datos, donde se almacena.

**Flujo alternativo:**

- El sistema comprueba la validez de los datos y, si los datos no son correctos, el mensaje es descartado y no es almacenado en la base de datos.
- Al utilizar nodos remotos, la obtención de los datos es mediante un proveedor mediante una API (el flujo no pasa por el servidor MQTT, accede directamente al servidor de Node-Red mediante comando HTTP GET).

**Pos condiciones:** Los datos son almacenados en el sistema (BDD).

## Anexo J. Código desarrollado a lo largo del trabajo

Debido a que el código realizado es muy extenso y la mayoría de funciones importantes ya se han comentado a lo largo del documento, el código lo enviaré mediante un enlace con lectura disponible para cualquier usuario del entorno UPC.

Accediendo al siguiente enlace:
https://drive.google.com/drive/folders/1SFI2ti2i9MlXyoXTpWJQWmgSW7fuDzJf?usp=sharing

Cualquier usuario del entorno "upc.edu" podrá visualizar la carpeta y descargar los archivos con el código utilizado en el desarrollo del proyecto.

**Modificaciones realizadas (protección de datos):**

En el objeto de especificaciones del servidor Node-RED, he modificado la línea que especifica la dirección IP donde se ha alojado el bróker del servidor MQTT.

```
{
    "id": "0fc4d345867ff802",
    "type": "mqtt-broker",
    "name": "localhost",
    "broker": "IP DE CONEXIÓN DEL BROKER",
    "port": "1883",
    "clientid": "",
    "autoConnect": true,
```

*Ilustración 51:* Rafael C. (2022). [Flow exportado Node-RED]

La inicialización de variables en el software del nodo físico requiere de el nombre del punto de acceso WiFi, IP y contraseña del router, de forma que he modificado estos parámetros con un String.

```
//Conexión a la red WiFi: nombre de la red y contraseña
#define WIFI_AP "Nombre del punto WiFi"
#define WIFI_PASSWORD "Contraseña del WiFi"

#define DHTPIN 4 //sensor DHT conectado en el pin 4
#define DHTTYPE DHT11 //mi sensor es el DHT11
const int MQpin = 2; // pin analógico 2
//declarar objeto:  sensor(pin: 2, sensor: DHT11)
DHT dht(DHTPIN, DHTTYPE);

//Nombre o IP del servidor mosquitto
// Establecer la IP igual a la IP de la red WiFi que estamos
char server[50] = "IP de la red WiFi"; // se puede consultar
// desde la terminal hay que ejecutar el comando: "ipconfig"
```

*Ilustración 52:* Rafael C. (2022). [Código modificado del software del nodo físico]

Para el desarrollo de la aplicación, adjunto una imagen de los paquetes y dependencias, junto con sus versiones:

```
dependencies:
  cupertino_icons: ^1.0.2
  firebase_auth: ^3.3.11
  firebase_core: ^1.10.6
  firebase_database: ^9.0.5
  flutter:
    sdk: flutter
  flutter_animation_progress_bar: ^2.0.1
  syncfusion_flutter_charts: ^19.4.53
  wave_progress_widget: ^0.0.1
```

*Ilustración 53:* Rafael C. (2022). [Paquetes y dependencias de la app]

## Anexo K. Cuadro de mando (interfaz de usuario mediante Node-Red)

Node-Red también nos ofrece la capacidad de gestionar una interfaz de usuario dinámica propia, en la que en nuestro caso, podemos presentar los datos obtenidos a tiempo real. La interfaz de usuario, no solo sirve para el muestreo y presentación de datos, también posee de nodos interactivos programables, los cuales podemos utilizar para aplicaciones muy diversas (dar permisos de administrador, emitir una respuesta hacia el exterior, modificar los datos a gestionar, etc.).

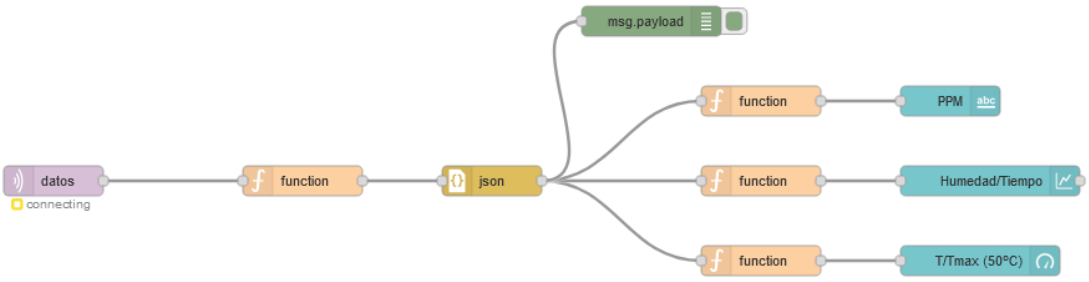


*Ilustración 54:* Rafael C. [Flujo Node-RED para la interfaz: nodo físico]

- Nodo MQTT in (copia del nodo MQTT del flujo físico, solo que esta vez no está nombrado)
- Nodo function: Función para añadir fecha en el msg.payload al String recibido del topic "datos" y dar formato JSON al String resultante.
- Nodo json (dejar por defecto): Para crear el objeto una vez dada la notación en el nodo anterior.
- Nodo function: filtramos el objeto para obtener el valor de los PPM dado en el payload.
- Nodo function: filtramos el objeto para obtener el valor de la Humedad dado en el payload.

- Nodo function: filtramos el objeto para obtener el valor de la Temperatura dado en el payload.
  - La estructura a seguir en el dashboard será la siguiente para asignar los datos: [Home] (tab): {Temperatura, Humedad, Calidad de aire} (groups)
- Nodo de texto: PPM: La representación de los PPM se mostrará en la interfaz de usuario como texto que se irá actualizando a medida que lleguen los datos, una vez asignado al grupo [Home] Calidad de aire.
- Nodo chart: Humedad/Tiempo: Mediante este nodo, tomaremos el valor de la Humedad y la representaremos en función de la hora en la que se recibe el dato, será asignado al grupo [Home] Humedad.
- Nodo gauge: T/Tmax (50ºC): Establecemos un valor de temperatura máxima de 50 en el rango y una mínima de 0, damos el valor de unidades y asignamos el grupo [Home] Temperatura.

Finalmente, si accedemos a la interfaz de usuario "*localhost:1880/ui*", podremos visualizarla, en este pequeño ejemplo de funcionamiento, se nos muestra lo siguiente:

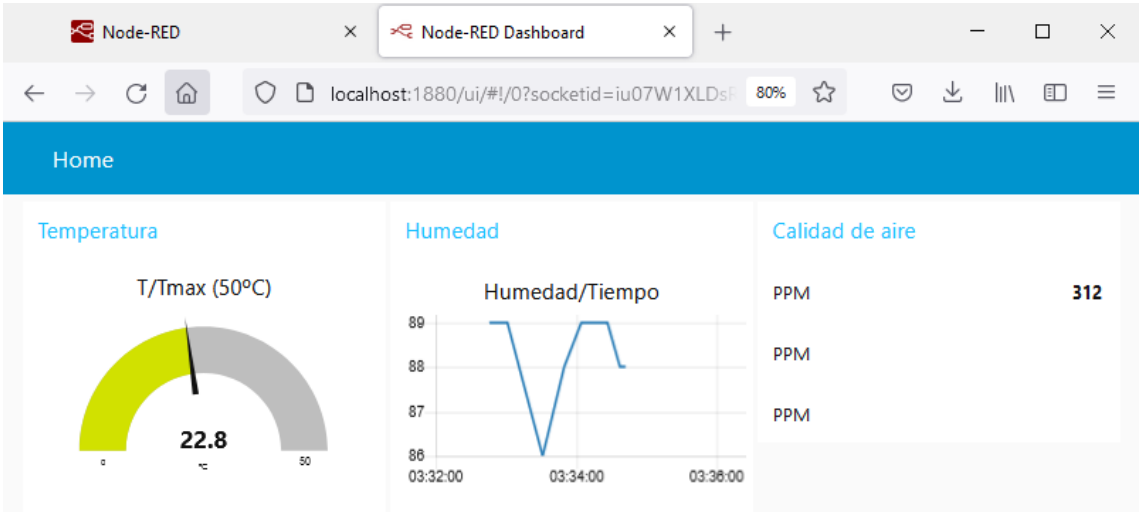


*Ilustración 55:* Rafael C. (2022) [Dashboard de Node-RED]