



# UNIVERSIDAD DE GRANADA

TRABAJO FIN DE GRADO  
GRADO EN INGENIERÍA INFORMÁTICA

## Diseño e implementación de una plataforma escalable para explotaciones agrarias

---

Utilizando ROS en el sector agrícola

**Autor**

Pedro Miguel Parrilla Navarro

**Directores**

Juan Manuel López Soler  
Juan José Ramos Muñoz



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍAS INFORMÁTICA Y DE  
TELECOMUNICACIÓN

—  
Granada, Noviembre de 2021



# **Diseño e implementación de una plataforma escalable para explotaciones agrarias**

Pedro Miguel Parrilla Navarro

**Palabras clave:** *Pub-Sub, agricultura, DDS, ROS, micro-ROS, Firebase, SQLite3, FastAPI, Angular*

## **Resumen**

En el sector agrícola está avanzando a pasos agigantados gracias a la inclusión de las diferentes tecnologías de comunicación, robóticas o todo lo relacionado con IoT. Las empresas están desarrollando e implementando sus plataformas en el mercado abarcando un gran número de clientes. En el documento se van a analizar los productos disponibles de manera genérica y tras esto realizar la propuesta de uno propio.

En este proyecto se va a diseñar e implementar una plataforma que utiliza comunicación un patrón Publicador-Suscriptor utilizando el conjunto de herramientas que proporciona ROS 2. Este trabaja sobre una capa DDS donde se implementa ese tipo de conexión. Tras la definición y conexión de los diferentes dispositivos, la información generada será almacenada en una base de datos para su posterior visualización en una interfaz web. Por último se realiza una evaluación de diferentes escenarios, añadiendo elementos de manera progresiva al hardware disponible.



---

## **Design and implementation of a scalable platform for agricultural farms**

Pedro Miguel Parrilla Navarro

**Keywords:** *Pub-Sub, agriculture, DDS, ROS, micro-ROS, Firebase, SQLite3, FastAPI, Angular*

### **Abstract**

The agricultural sector is advancing very fast due to the use of different communication technologies, robotics and everything related to IoT. Companies are developing and implementing their platforms in the market covering a large number of customers. In this document we are going to analyse the available products in a generic way and after this we are going to make a proposal for one of our own.

In this project we are going to design and implement a platform that uses a Publisher-Subscriber communication pattern using the set of tools provided by ROS 2. This works on a DDS layer where this type of connection is implemented. After the definition and connection of the different devices, the information generated will be stored in a database for later visualisation in a web interface. Finally, an evaluation of different scenarios is carried out, adding elements progressively to the available hardware.



---

D. **Juan Manuel López Soler**, Profesor del Departamento de la Teoría de la Señal, Telemática y Comunicaciones, y D. **Juan José Ramos Muñoz**, Profesor del Departamento de la Teoría de la Señal, Telemática y Comunicaciones

**Informo:**

Que el presente trabajo, titulado *Diseño e implementación de una plataforma escalable para explotaciones agrarias*, ha sido realizado bajo nuestra supervisión por **Pedro Miguel Parrilla Navarro**, y autorizo la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a Junio de 2018.

**El/la director(a)/es:**

**Juan Manuel López Soler**

**Juan José Ramos Muñoz**



# Agradecimientos

Gracias a todos aquellos profesores que me han marcado y ayudado durante mi etapa de estudiante.

Gracias a mis amigos y compañeros de piso quienes me han acompañado durante esta etapa universitaria, tanto en las buenas, como en las malas.

Gracias a Alex por estar siempre ahí cuando necesito que me echen una mano.

Gracias a MJ por ayudarme y cuidarme durante estos meses de trabajo.

Gracias a mis padres y a mi hermano por haberme apoyado, orientado y hasta regañado en los momentos necesarios.

Y por último volver a destacar las gracias a mi padre, ya que sin él seguramente este proyecto no habría sido abordado de la misma manera.



# Índice general

<b>1. Introducción</b>	<b>17</b>
1.1. Descripción del problema . . . . .	17
1.2. Motivación . . . . .	17
1.3. Objetivos . . . . .	18
1.4. Estructura del documento . . . . .	18
<b>2. Estado del arte</b>	<b>19</b>
2.1. Agricultura inteligente . . . . .	19
2.1.1. Robots en el sector . . . . .	19
2.2. Sistemas utilizados en la industria . . . . .	20
2.2.1. Despliegues centralizados . . . . .	23
2.2.2. Empresas con despliegues descentralizados . . . . .	24
2.3. Tecnologías para publicación suscripción . . . . .	24
2.3.1. RabbitMQ . . . . .	25
2.3.2. Apache Kafka . . . . .	25
2.3.3. ROS2 . . . . .	26
<b>3. Planificación</b>	<b>29</b>
3.1. Metodología utilizada . . . . .	29
3.2. Temporización . . . . .	29
3.3. Presupuesto . . . . .	31
3.4. Tiempo total empleado . . . . .	32
<b>4. Diseño de la propuesta</b>	<b>35</b>
4.1. Descripción del entorno y análisis de requisitos . . . . .	35
4.1.1. Sensores . . . . .	35
4.1.2. Actuadores . . . . .	36
4.1.3. Controladores . . . . .	36
4.1.4. Interfaces de usuario . . . . .	36
4.2. Requisitos de la plataforma a desplegar . . . . .	37
4.3. Diseño general del proyecto . . . . .	37
4.4. Diseño de clases de cada dispositivo . . . . .	39
4.5. Diseño de interfaces en ROS2 . . . . .	40

---

## *ÍNDICE GENERAL*

4.6. Diseño de la Base de Datos . . . . .	43
4.7. Diseño de la interfaz web . . . . .	44
<b>5. Implementación . . . . .</b>	<b>47</b>
5.1. ROS 2 . . . . .	47
5.1.1. Creación de las diferentes interfaces utilizadas en los tópicos	48
5.1.2. Desarrollo de los nodos . . . . .	49
5.1.3. Micro ROS en microcontroladores . . . . .	54
5.1.4. Conexión a la base de datos . . . . .	56
5.1.5. Simulación de un entorno con diferentes nodos . . . . .	57
5.2. API REST para obtención de datos . . . . .	58
5.3. Interfaz Web . . . . .	59
5.3.1. Home . . . . .	59
5.3.2. Tabla de datos . . . . .	60
5.3.3. Gráficos . . . . .	61
5.3.4. Mapa . . . . .	62
5.3.5. Login . . . . .	63
<b>6. Evaluación . . . . .</b>	<b>65</b>
6.1. Escenario 1 . . . . .	65
6.2. Escenario 2 . . . . .	66
6.3. Escenario 3 . . . . .	67
<b>7. Conclusiones y trabajo futuro . . . . .</b>	<b>71</b>
7.1. Trabajo futuro . . . . .	71
<b>Acónimos . . . . .</b>	<b>77</b>

# Índice de figuras

2.1. Imágenes de diferentes actuadores . . . . .	22
2.2. Despliegue centralizado (Elaboración propia) . . . . .	23
2.3. Datalogger GPRS junto a un sensor (Imagen de Nazaries [21]) . . . . .	24
2.4. OMG DDS interoperabilidad . . . . .	26
2.5. Capas disponibles con ROS 2 . . . . .	27
2.6. Comparación estructura ROS2 y microROS . . . . .	28
3.1. Diagrama de Gantt previo . . . . .	30
3.2. Presupuesto del proyecto . . . . .	31
3.3. Resumen de horas obtenido de Clockify juntando todas las diferentes tareas . . . . .	32
3.4. Diagrama de Gantt . . . . .	33
4.1. Dibujo de un invernadero tipo capilla (Novagric) . . . . .	38
4.2. Diagrama de un invernadero con vista aérea . . . . .	38
4.3. Diagrama de clases . . . . .	39
4.4. Estructuras definidas utilizadas en los tópicos . . . . .	40
4.5. Interfaces . . . . .	41
4.6. Diagrama de comunicación entre nodos . . . . .	42
4.7. Tabla de Temperatura . . . . .	43
4.8. Página principal . . . . .	44
4.9. Tablas de datos . . . . .	44
4.10. Gráficas . . . . .	45
4.11. Mapa para visualizar la localización de los componentes . . . . .	46
5.1. Ejecución de los diferentes nodos con parámetros . . . . .	53
5.2. Herramientas con el comando ros2 . . . . .	53
5.3. Gráfico generado con rqt-graph de la red . . . . .	54
5.4. Montaje ESP32 con DHT11 . . . . .	56
5.5. Simulación del escenario en la red . . . . .	58
5.6. Página principal . . . . .	60
5.7. Tablas de datos . . . . .	61
5.8. Gráficas de radiación . . . . .	61
5.9. Mapa interactivo con los diferentes sensores . . . . .	62

## ÍNDICE DE FIGURAS

---

5.10. Login implementado . . . . .	63
6.1. Escenario 1 . . . . .	66
6.2. Escenario 2 . . . . .	67
6.3. Escenario 3 . . . . .	68
6.4. Tabla de Temperatura . . . . .	68
6.5. Red en el Escenario 3 . . . . .	69
6.6. Gráfica generada por Wireshark . . . . .	70

# Listings

5.1.	Estructura del directorio de trabajo . . . . .	48
5.2.	Código FloatDataNode.msg . . . . .	48
5.3.	Código UploadFile.srv . . . . .	48
5.4.	Comando para la creación de un paquete de Python con ROS 2	49
5.5.	Estructura del directorio de desarrollo con Python . . . . .	49
5.6.	Plantilla genérica para la creación de nodos . . . . .	50
5.7.	Publisher del sensor de temperatura . . . . .	51
5.8.	Subscriber del actuador . . . . .	51
5.9.	Creación de cliente de ROS2 en Python . . . . .	52
5.10.	Publisher Timer y Executor en C para microcontrolador . . . . .	55
5.11.	Ejecución Agente de micro-ROS en un contenedor Docker . . . . .	56



# Capítulo 1

## Introducción

En este Trabajo Fin de Grado se va a diseñar, implementar y evaluar una plataforma basada en el paradigma Publicador-Suscriptor [7] para un caso de uso en agricultura, utilizando el sistema Robot Operating System (ROS) [2]. Esto es un conjunto de librerías y herramientas para creación de aplicaciones orientadas a creación de robots. En esta plataforma se identifican dispositivos muy variados divididos en dos grupos genéricos: sensores y actuadores. Además, se contará con un controlador que recopile la información y un gestor de la base de datos para almacenarlo. Por último se dispondrá de una interfaz web para la visualización de los datos más relevantes. A destacar, se utilizarán diferentes dispositivos hardware para realizar el despliegue y poder simular algunos elementos de un escenario real.

### 1.1. Descripción del problema

El problema viene en que los agricultores necesitan saber el estado del terreno o de la plantación durante el periodo de trabajo para obtener el máximo rendimiento posible sin pérdidas en la cosecha. Por ello en el caso de que las condiciones climatológicas cambien o se origine una plaga, tienen que responder a estas acciones de una forma u otra.

En algunos casos las propias máquinas ya lo tienen automatizado, pero hasta donde el autor conoce muy pocos tienen una interfaz o un tipo de respuesta que les confirme que este trabajo ha sido realizado y que los trabajadores puedan consultarla desde sus casas. Por lo tanto la mayoría de ellos acaban acudiendo al terreno a comprobar este correcto funcionamiento.

### 1.2. Motivación

Esta temática de trabajo ha sido seleccionada ya que el sector de la tecnología aplicada en la agricultura está en auge. Han surgido empresas con productos muy variados estos últimos años, ofreciendo diferentes servicios según el interés de los clientes.

Como el crecimiento y las novedades en la tecnología aumentan de manera exponencial, algunos de estas instalaciones pueden quedarse obsoletas en poco tiempo según las herramientas utilizadas.

Por ello se ha escogido este tema, para proponer un proyecto el cual utiliza herramientas modernas para transmitir, automatizar y almacenar una serie de datos de manera digital, gracias a las Tecnologías de Información y Comunicación (TICs).

### **1.3. Objetivos**

Vamos a destacar en principio tres objetivos a cumplir, y tras esto una breve explicación de ellos.

En principio, se va a realizar un diseño que permita monitorizar y automatizar procedimientos en una explotación agrícola.

El siguiente es la posibilidad añadir nuevos tipos de sensores o actuadores, sin afectar poco o nada a los demás componentes de este diseño. Un ejemplo sería un robot para transporte o un dron que realice un recorrido mientras vierte un producto.

Por último almacenar estos datos, para luego poder tratarlos y procesarlos para su visualización. Por destacar, en un trabajo futuro podrían ser utilizados para el entrenamiento de alguna Inteligencia Artificial (IA).

### **1.4. Estructura del documento**

En el capítulo 2 se va a abordar el estado del sector a tratar, las tecnologías aplicadas por las diferentes empresas y un listado de las diferentes herramientas posibles a utilizar en el proyecto.

En el capítulo 3 se habla sobre la metodología aplicada, como ha sido llevado a cabo el proyecto con su respectivo diagrama de Gantt y un esquema del presupuesto total.

El capítulo 4 analiza el proyecto, que es necesario desarrollar y en qué partes se va a dividir.

En el capítulo 5 se explica la implementación realizada del proyecto, siguiendo el orden y estructura establecida en la parte del análisis.

El capítulo 6 presenta diferentes escenarios evaluados, describiendo que componentes se utilizan y midiendo la memoria al ejecutar ciertos procesos.

Por último, en el capítulo 7 se hace una reflexión sobre todo lo aprendido y proyectado en este trabajo, obteniendo conclusiones sobre ello. Tras esto se finaliza destacando algunas tareas a realizar en el futuro.

## Capítulo 2

# Estado del arte

En este capítulo se revisarán la literatura relacionada con la Agricultura inteligente, así como las plataformas comerciales existentes.

### 2.1. Agricultura inteligente

La agricultura inteligente [4], o agricultura 4.0 se basa en la recopilación y análisis de datos sobre el campo, con el objetivo de mejorar la calidad de los cultivos y reducir las consecuencias en el medio ambiente.

Esto es posible con el uso de las nuevas tecnologías. Los diferentes robots, sensores, software, etc, son capaces de realizar tareas agrícolas en menos tiempo que el ser humano y con mejor resultados.

Ejemplos de tecnología utilizada en la agricultura:

- **Sensorización ambiental**, referente al uso de sensores, obteniendo la información en tiempo real para optimizar mejor los recursos.
- **Uso de drones y teledetección**, con el objetivo de tomar fotos para analizar el estado, o incluso pulverizar un producto en cultivos.
- **Sistemas predictivos**, permitiendo la anticipación a las diferentes condiciones meteorológicas en las próximas horas o días.
- **Inteligencia artificial**, para tomar decisiones o realizar recomendaciones en base a los datos procesados.

A parte de estos puntos, podemos destacar unos de los temas tocados en este proyecto, la robotización.

#### 2.1.1. Robots en el sector

En [3] hay un artículo donde se comenta que los desarrolladores actualmente están trabajando en máquinas autónomas capaces de realizar tareas como las siguientes:

- **Desbrozar.** Eliminar las malas hierbas mediante el reconocimiento de imagen con una cámara y expulsar dosis de herbicida según la mala hierba y el tamaño
- **Recogedor.** Realizar una recolecta del cultivo mediante la utilización de brazos robóticos, analizando la madurez de este.
- **Abonador.** Analiza el tamaño y color de las hojas de cada planta, aplicando una dosis de fertilizante según la necesidad de este.

Todas estas máquinas se podrían comunicar entre ellas, siendo controladas por un operador o trabajando de manera autónoma con ciertos horarios. Estas podrían ser utilizadas en los siguientes escenarios que se explicarán a lo largo del documento.

## 2.2. Sistemas utilizados en la industria

Esta parte ha sido una de las más complicadas de escribir, ya que no es fácil saber qué tecnologías utilizan estas empresas debido a que las explicaciones realizadas en sus respectivas páginas web son muy genéricas del estilo: *Consulta tus datos en tiempo real desde cualquier dispositivo*.

Como se ha visto en el anterior apartado, la mayoría de empresas tienen en común: **sensores y actuadores**, los cuales se van a agrupar de ahora en adelante como **nodos**.

Estos son la base del sistema, ya que son los **productores** de los datos a tratar, y de los cuales se obtiene la mayoría de la información. A continuación voy a realizar una lista de los algunos tipos que se deberían contemplar en el diseño, pudiendo ser estos aún más según las necesidades:

- Temperatura del suelo, para ver el estado de la planta
- Humedad, tanto fuera como dentro del invernadero
- Estado hídrico del cultivo, para controlar los patrones de riego
- Velocidad del Viento
- Dirección del viento

Estos productores sirven para que los actuadores hagan ciertas funciones. Pero además hay otros actuadores que quizás no necesitan la información de estos sensores como un robot de transporte, brazo robot de recolecta, o algo más factible como los drones para realización de fotografías. Algunos actuadores que si utilizan la información de los productores son:

- **Ventanas**, para transpirar y cambiar el aire de dentro

## 2.2. SISTEMAS UTILIZADOS EN LA INDUSTRIA

- **Calefactores**, para aumentar la temperatura
- **Humidificadores**, necesario en climas secos
- **Tuberías para riego**, permitiendo la circulación del agua
- **Pantallas térmicas**, con el objetivo de mantener fresco el terreno en días de mucho calor
- **Destratificadores**, ciclando el aire del interior, ya que el aire frío suele quedarse a nivel del suelo y el caliente tiende a subir.

En el siguiente conjunto de figuras 2.1 hay mostrados algunos de los actuadores mencionados



(a) Ventanas (Imagen de la web de agroprecios)



(b) Pantallas térmicas (Imagen de la web fertri.com [13])



(c) Destratificador (Imagen de la web de Alarcontrol [27])

Figura 2.1: Imágenes de diferentes actuadores

## 2.2. SISTEMAS UTILIZADOS EN LA INDUSTRIA

Por lo tanto, tras estudiar los diferentes productos desplegados por las empresas que las empresas utilizan, el autor de este trabajo propone dividir las soluciones actuales en dos grupos: **centralizadas o descentralizadas**.

### 2.2.1. Despliegues centralizados

En este caso, todo está conectado a un nodo que orquesta toda la plataforma. Un ejemplo de empresa que utiliza este despliegue es Alarcontrol S.L. [27].

La figura 2.2 nos da una idea de cómo es esta implementación. Esta consta de un panel central, donde un dispositivo gestiona todas las tareas, recopilando los datos de los diferentes sensores colocados en puntos estratégicos del terreno y conectados cada uno mediante un cable a la placa central.

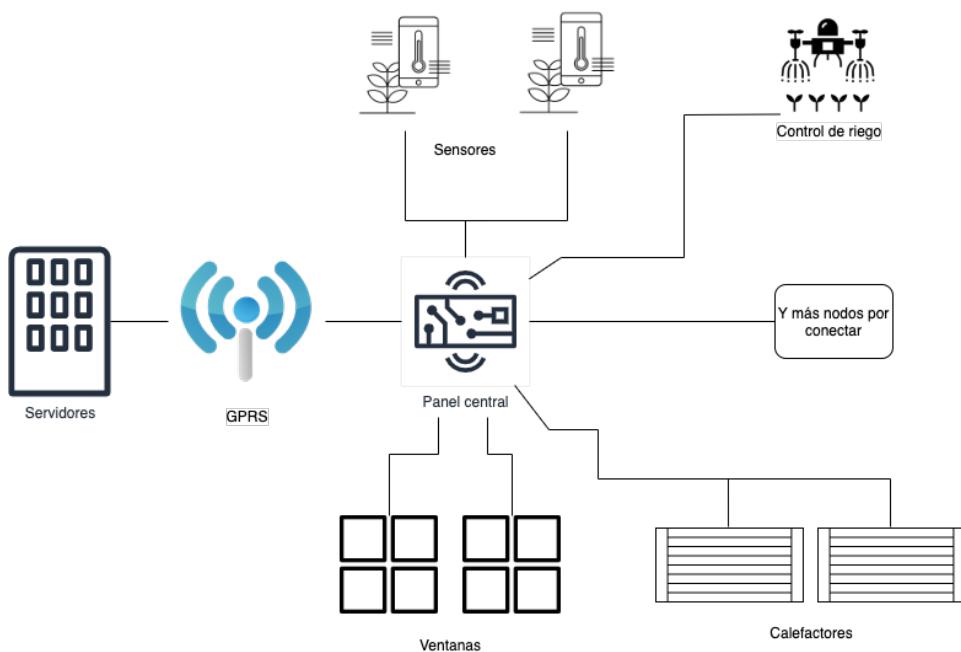


Figura 2.2: Despliegue centralizado (Elaboración propia)

En el panel central se podrían encontrar diferentes botones para diversas acciones, además de una réplica de este dentro del invernadero para gestionar otros actuadores más cercanos, como ventanas o calefactores.

Por último tiene conexión Servicio General de Paquetes vía Radio (GPRS). Esta la utiliza para transmitir los datos a un servidor central y que el usuario desde el móvil pueda visualizarlos.

Una posible desventaja de este servicio es la disponibilidad, ya que en el caso de que falle el gestor central no se podría disponer de ningún dato generado ni de los actuadores. Esto es debido a que no hay un sistema de respaldo, por lo tanto por muchas pruebas que se le hayan realizado, podría fallar.

### 2.2.2. Empresas con despliegues descentralizados

A diferencia del anterior, al ser descentralizado, no todo es gestionado por un único dispositivo, si no que podemos encontrar varios. Ejemplos de empresas que utilizan este diseño son Nazaries [21] y Hortisys [17].

Este sistema distribuye las tareas en más nodos, que poseen conectores genéricos para enchufar diferentes dispositivos, en función requiera el usuario. En el caso de la empresa Nazaries, los nombra como *Dataloggers*, que viene a ser recopilador de datos. Un ejemplo de este dispositivo lo podemos visualizar en la siguiente figura:



Figura 2.3: Datalogger GPRS junto a un sensor (Imagen de Nazaries [21])

Como podemos observar, en el pie de la figura se hace referencia de GPRS. Esto es debido a que hay diferentes tipos de recopiladores de información según su conexión o visualización de datos: bluetooth, sigfox [11] o mostrados en un terminal.

Esto significa que trabaja de manera similar al despliegue centralizado, lo único que divide la carga, ya que diferentes nodos se encargan de recopilar esta información.

Pero esto también plantea un problema, y es que si uno de esos nodos se rompe, todos los datos que recopilaba no pueden ser visualizados.

Este supuesto fallo sería menos grave que en el caso anterior, ya que en el otro la plataforma entera dejaría de funcionar, y en este conlleva la pérdida de información del número de sensores conectados a este *datalogger*.

## 2.3. Tecnologías para publicación suscripción

En el artículo de la bibliografía *Meeting iot platforms re-requirements with open pub/sub solutions* [7], se define el patrón pub-sub como un Message-oriented Middleware (MoM) que proporciona una comunicación distribuida, asíncrona entre productores y consumidores de mensajes. Este patrón presenta tres tipos principales de desacoplamiento que lo hacen especialmente adecuado para los despliegues de Internet of the Things (IoT) a gran escala:

## 2.3. TECNOLOGÍAS PARA PUBLICACIÓN SUSCRIPCIÓN

- Los productores de mensajes (publicadores) y los consumidores (suscriptores) están desacoplados en el tiempo, es decir, no tienen que estar conectados al mismo tiempo.
- Los mensajes no se dirigen a un consumidor en específico, si no a una dirección simbólica (canal, tópico)
- La mensajería es asíncrona y no se bloquea.

Uno de los elementos fundamentales de los sistemas pub/sub es la correspondencia entre productores y suscriptores, que puede basarse en distintos tipos de filtrado, principalmente de temas o contenidos. Este filtrado suele ser realizado por múltiples intermediarios de mensajes dedicados.

Tras la introducción de este sistema, se va a realizar una breve descripción de algunas tecnologías que lo aplican y cuál ha sido escogida para este proyecto.

### 2.3.1. RabbitMQ

RabbitMQ [24] es un software message-broker (negociador de mensajes) originalmente implementado con el protocolo Advanced Message Queuing Protocol (AMQP), con soporte a otros protocolos como por ejemplo Streaming Text Oriented Messaging Protocol (STOMP) y MQ Telemetry Transport (MQTT). Tiene la ventaja de que es independiente al lenguaje utilizado, y puede ser desplegado en .NET, Python, PHP, Ruby, etc.

Esta tecnología empezó como sistema de colas, por ello la parte de MQ en el nombre. Debido a esto, la implementación que realiza pub-sub está construida sobre este sistema. Esto puede ser que no sea lo más eficiente respecto a otra alternativa que su diseño si sea basado en el modelo de pub-sub. Además utiliza brokers (intermediarios entre pub-sub), haciendo que la conexión no sea directa y surja otro punto de posible fallo.

### 2.3.2. Apache Kafka

Kafka [18] es un sistema de mensajería tipo pub-sub, escrito en lenguaje Scala, siendo escalable, duradero y tolerante a fallos. Es utilizado por empresas como Linkedin, Yahoo, Twitter y otras.

Su principal uso destaca en el análisis a tiempo real, pero se puede utilizar para supervisión, reproducción de mensajes, agregación de registros, recuperación de errores y el seguimiento de la actividad de un sitio web.

Pero para uso en IoT donde los dispositivos están conectados a un centro de datos o a la nube, Kafka no es idónea ya que necesita bastante configuración. Además hay una serie de puntos que no la hacen idonea para este entorno:

- No soporta una enorme cantidad de tópicos
- Los brokers necesitan estar directamente direccionados por los clientes

- Algunas de las claves de las funcionalidades de IoT no están presentadas en Kafka, como por ejemplo *keep alive*. Esta hace referencia al historial de datos a mantener en la red.
- Es necesario una conexión estable TCP, no muy presente en estos entornos, haciendo que los dispositivos gasten recursos en la reconexión.

### 2.3.3. ROS2

ROS2 [2] es un conjunto de herramientas software utilizadas principalmente para la creación de estos robots. Empresas como BMW de automoción o la NASA utilizan ROS en algunos de sus proyectos. Por destacar, es utilizado en el Robonaut 2 [25].

Las características principales que surgieron tras su desarrollo fueron:

- Sistemas en tiempo real, para diferentes tratamientos de datos
- Equipos de múltiples robots, pudiendo comunicarse entre ellos para diferentes tareas
- Para conexiones no ideales, es decir, gestionando perdidas o delay de una red WiFi de poca calidad

Por defecto, ROS2 utiliza Data Distribution Service (DDS) [23] como middleware para la comunicación. DDS está orientado a usarse en computación distribuida implementando el patrón de esta sección, publicador-subscriptor. Permite el manejo de los mensajes de manera transparente sin la necesidad de la intervención del usuario, incluyendo algunas funciones como quien debería recibir el mensaje, donde están situados los consumidores o qué pasaría si el mensaje no pudiese ser enviado. Además DDS soporta diferentes lenguajes como C, C++, Java o Python gracias a la Application Programming Interfaces (APIs) disponibles. Esto permite que diferentes dispositivos, con sistemas operativos o lenguajes distintos puedan comunicarse. Esto viene a ser conocido como interoperabilidad.

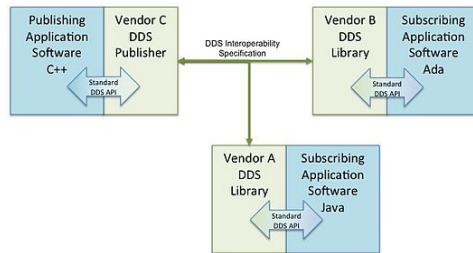


Figura 2.4: OMG DDS interoperabilidad

El escenario mostrado en la figura 2.4 se puede obtener del artículo [23]. En él se explica la interoperabilidad del protocolo utilizado para la conexión de los diferentes dispositivos.

### 2.3. TECNOLOGÍAS PARA PUBLICACIÓN SUSCRIPCIÓN

Tras explicar algunos conceptos y funcionalidades sobre DDS, caracterizar que no existen brokers en algunas de sus implementaciones, ya que utiliza un protocolo multicast basado en descubrimiento de dispositivos (intermediarios entre pub-sub). Además simplifica el despliegue, minimiza la latencia, maximiza la escalabilidad y aumenta la fiabilidad. Por tanto es idóneo para aplicaciones IoT que requieren de una arquitectura duradera y fiable.

Respecto a ROS 2, utiliza un middleware para ser independiente a la implementación de DDS utilizada, además de ofrecer una API para los diferentes lenguajes de programación soportados. Esta información se puede visualizar en la siguiente imagen:

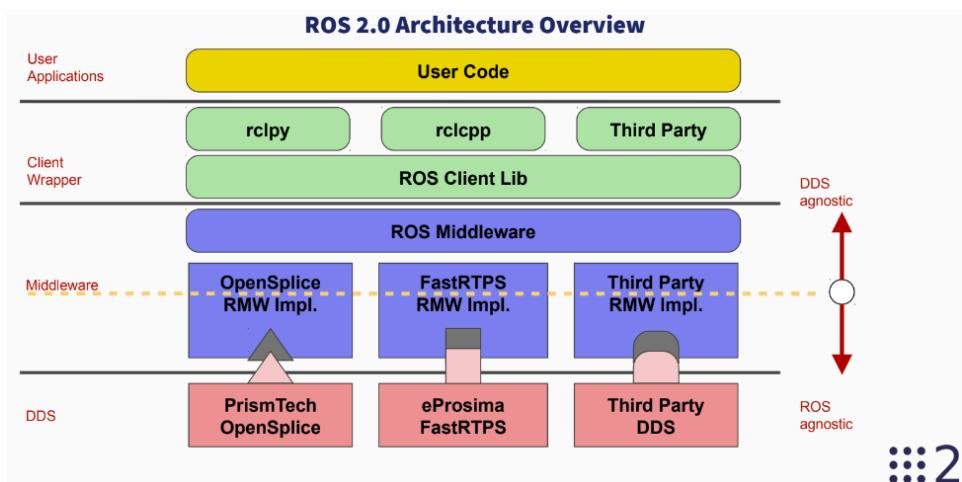


Figura 2.5: Capas disponibles con ROS 2

En la figura 2.5 se puede observar las capas de la API que ofrece ROS 2.

Por último, ROS2 es compatible con múltiples implementaciones de DDS, como de eProxima's Fast DDS [10], RTI's Connext DDS [26] y Eclipse Cyclone DDS [9]. Por defecto, ROS2 utiliza el producto de eProxima's.

Cabe destacar que para poder utilizar ROS en microcontroladores, es necesario el uso de microROS. Este se diferencia de que la implementación es realizada en C/C++, además que deberá situarse en una capa por encima que actúe como sistema operativo, como Real-time operating system for microcontrollers (FreeRTOS).

El diagrama de comunicación y de capas se puede visualizar en la siguiente imagen:

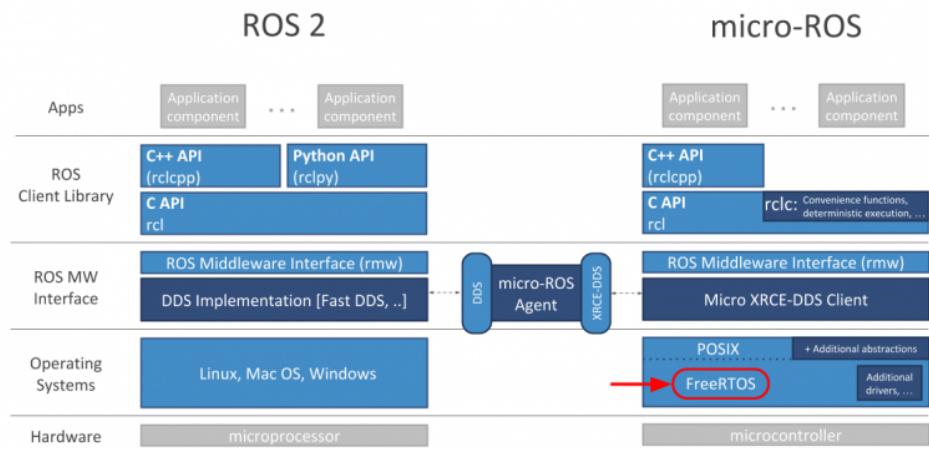


Figura 2.6: Comparación estructura ROS2 y microROS

Como se puede observar en 2.6, en el centro esta situado una figura que pone micro-ROS Agent. Este es el encargado de comunicar la parte de DDS (la parte del middleware de ROS) generada por el microcontrolador con las diferentes implementaciones de DDS que existan en la red.

# Capítulo 3

## Planificación

En este capítulo se describe la metodología utilizada en el proyecto, cómo se van a gestionar las tareas a lo largo del tiempo con un diagrama de Gantt y tras esto la evaluación del coste de todo el material y trabajo a realizar.

Por último se planteará otro diagrama de Gantt donde se muestran las franjas de las actividades de cómo han ocurrido de manera real y si son próximas a las estimadas.

### 3.1. Metodología utilizada

Respecto a la metodología utilizada va a ser una variante de Scrum, ya que se irán realizando tareas (sprints) que serán evaluadas en períodos semanales por el tutor. Estas tienen el objetivo de ir expandiendo el producto poco a poco, y se puedan visualizar los diferentes requisitos aplicados.

Tras la finalización de cada tareas, hay una reunión donde se discute la tarea realizada, y en el caso de cómo ha ido su desarrollo, se volverá a revisar o se avanzará a la siguiente tarea, pudiendo ir modificando la orientación del proyecto.

Además se ha hecho uso de un repositorio Github para realizar un control del siguiente proyecto. El link es el siguiente: [github.com/parrilla/ROS2\\_TFG](https://github.com/parrilla/ROS2_TFG)

### 3.2. Temporización

Para el registro de horas se va a utilizar la aplicación Clockify [6]. Esta permite ir creando tareas diarias con un nombre, un tag referente a que grupo pertenece e indicar el proyecto. Este registro de horas es necesario para ir realizando comparaciones con el tiempo que se ha estimado para cada tarea.

A fecha de la planificación del proyecto se estima que con las horas llevadas a cabo, este durará unas 350-400h aproximadamente, dependiendo de si se han estimado bien el tiempo dedicado a cada tarea.

Por destacar, indicar que la temática llevaba establecida desde Abril de 2021, pero debido a que no se pudo empezar por cuestiones académicas, se ha acabado

trasladando a estas fechas.

En el siguiente diagrama de Gantt 3.4 se detallan las diferentes tareas a realizar:

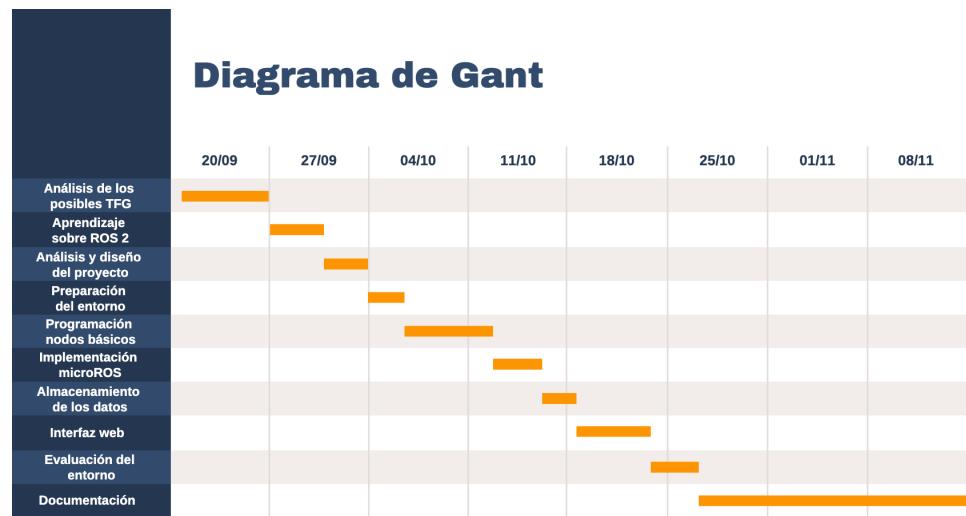


Figura 3.1: Diagrama de Gantt previo

Se van a explicar las diferentes categorías mostradas en el diagrama y que actividades abarcan cada una:

- **Análisis de los posibles TFG.** Engloba la parte previa a la realización del proyecto, búsqueda de diferentes temáticas, información sobre las tecnologías a utilizar y casos de uso reales sobre este.
- **Aprendizaje sobre ROS 2.** Se agrupa el tiempo de aprendizaje de la tecnología y los conceptos relacionados.
- **Ánalisis y diseño del proyecto.** Una vez escogida la base del proyecto se establece la estructura del proyecto, elementos necesarios para su desarrollo tanto hardware y software, reestructuración de diferentes partes implementadas y derivados.
- **Preparación del entorno.** En los dispositivos utilizados es necesario la instalación y creación de un entorno de trabajo.
- **Programación nodos básicos.** Hace referencia a las implementaciones realizadas para los nodos principales a utilizar más el testeo de su funcionamiento.
- **Implementación microROS.** Para los microcontroladores utilizados, es necesario el uso de esta tecnología ya que no poseen un gran rendimiento para estas tareas.

### 3.3. PRESUPUESTO

- **Almacenamiento de los datos.** Toda la información generada en la red es almacenada en una base de datos.
- **Interfaz web.** Creación de una interfaz para mostrar los diferentes datos mencionados.
- **Evaluación del entorno.** Ejecución de diferentes nodos de manera progresiva con el hardware disponible.
- **Documentación.** Descripción de todo el trabajo realizado, recogido en este documento.

### 3.3. Presupuesto

En esta sección se van a tratar los costes de la realización de este proyecto, incluyendo tanto el hardware como el software, además de considerar las horas en su elaboración.

Función	Marca	Modelo	Descripción	Cantidad	Precio unitario	Precio total
<b>Hardware</b>						
Portátil	Asus	GL752V	CPU : i7 RAM: 24 GB Uso 2 meses	1	1200€ durante 4 años de uso utilizado 2 meses	50
Raspberry Pi 3B	Raspberry	3B	CPU: Core 2 Quad RAM: 1GB Con carcasa y alimentación	1	55€	55€
Micro controlador	ESP	ESP32		2	7,99€	15,98€
Sensores	KY-015	DHT-11	Sensor temperatura y humedad	2	3€	6€
<b>Software</b>						
Comunicación entre nodos	ROS2		Software libre	1	0€	0€
Interfaz web	Angular		Software libre	1	0€	0€
Base de Datos	Firebase		Cobra por número de lecturas realizadas. No se ha superado el límite	1	0€	0€
IDE	VS Codium		Software libre	1	0€	0€
Gestor de contenedores	Docker		Software libre	1	0€	0€
<b>Mano de obra</b>						
Salario medio Ingeniero Informático por horas				400	13,05€	5.220,00€
				Total	<b>5346,98</b>	

Figura 3.2: Presupuesto del proyecto

El precio del portátil se ha hecho una estimación de la vida útil, con una media de 4 años, y como se ha utilizado durante 2 meses, el precio correspondiente es ese.

Por lo tanto, el gasto final será de 5346,98€ sumando las diferentes partes. A destacar del gráfico, el presupuesto de ingeniero informático ha sido obtenido de la web Talent.com [29], con una media de 13,05€ la hora.

Hay que tener en cuenta que el escenario mayormente será virtual ya que respecto al hardware se van a utilizar diferentes elementos disponibles por casa. Si se tuviese que realizar un presupuesto real sería necesario contactar con diferentes personas con conocimientos relacionados a los sensores del mercado, cableado necesario, elementos para la comunicación, placas a utilizar y estancia de la web en un servidor.

En este caso respecto al hardware se va a utilizar un portátil para el desarrollo y despliegue, una Raspberry Pi 3B con Ubuntu Server y dos placas ESP32 con conectividad wifi junto a sensores DHT-11, cuyo coste se ve reflejado en la tabla anterior 3.2.

### 3.4. Tiempo total empleado

En la figura 3.3 se pueden visualizar la distribución de horas final recogidas en la aplicación Clockify.

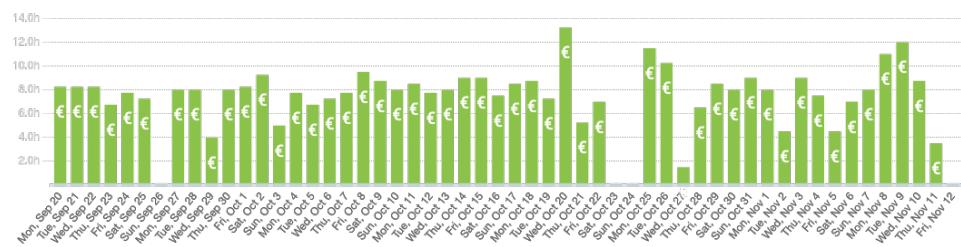


Figura 3.3: Resumen de horas obtenido de Clockify juntando todas las diferentes tareas

A fecha de la escritura de esta sección, el proyecto ha tomado unas 417h en realizarse, desde la búsqueda de que proyecto a realizar hasta la parte de documentación tras su finalización.

A continuación se adjunta el diagrama de Gantt tras apuntar las tareas realizadas:

### 3.4. TIEMPO TOTAL EMPLEADO

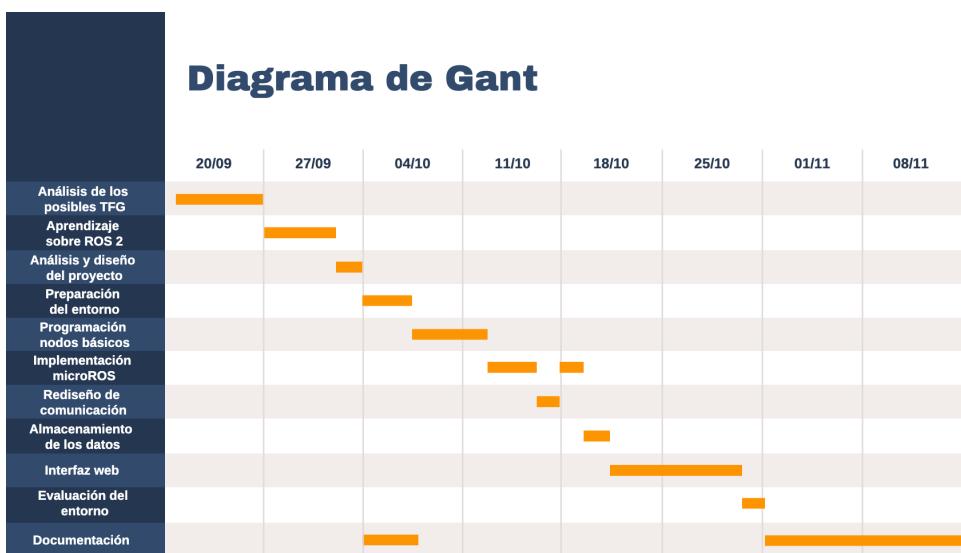


Figura 3.4: Diagrama de Gantt

A destacar, la tarea del rediseño de comunicación que es la nueva. Esto fue debido a que el tipo de mensaje enviado necesitaba más información de la que se había añadido al principio.



## **Capítulo 4**

# **Diseño de la propuesta**

En este capítulo se va a analizar los requisitos que tiene que tener este proyecto, que se va a abarcar en este trabajo, como va a ser la red de dispositivos que es necesario desplegar, de qué tipo son e identificar qué tarea va a realizar cada uno de estos nodos.

Para el análisis del escenario que se supone el contacto con un cliente propietario de un invernadero, obteniendo las medidas y el número de nodos necesarios. A partir de ahí se puede realizar un análisis de qué es necesario para el despliegue del producto.

Con esta breve introducción, se van a identificar los requisitos que debe cumplir el proyecto. Tras esto se realizarán diferentes diseños para entender la estructura del código y de la red.

### **4.1. Descripción del entorno y análisis de requisitos**

Para empezar, se va a suponer que tenemos un invernadero dividido en diferentes secciones. Estas se establecen en función del radio de actuación de algunos sensores o actuadores. Por lo tanto en cada una debe haber diferentes nodos que son necesarios para cumplir las expectativas y requisitos. Los dispositivos a desplegar se pueden agrupar en los siguientes apartados:

#### **4.1.1. Sensores**

En este proyecto y en la mayoría, los datos son la base de todo, sin ellos no se puede trabajar. Por ello se encuentran presentes diferentes generadores de estos datos, pudiendo ser como se ha comentado antes, sensores de temperatura, de humedad, de radiación, de la dirección del viento o velocidad, y algunos más.

Cada usuario necesita un tipo de estos productores según el cultivo que vaya a sembrar o la localidad del terreno. Además también hay que tener en cuenta el presupuesto, ya que según la calidad del sensor o el número de sensores que se requieran, hará que aumente el costo.

Dependiendo del tipo de cultivo, será necesario tener una mayor diversidad de dispositivos recopilando información. Hay más factores que influyen en este número, como el terreno en función de la cantidad de metros que disponga, la localización e incluso la inclinación de este. Se destacan estas últimas dos características ya que hay zonas donde el clima varía mucho entre el día y la noche.

Por tanto como se ha mencionado como requisito, es necesario poder aumentar el número de estos productores sin tener que tocar todo el sistema desplegado.

#### 4.1.2. Actuadores

En el caso de tener productores, también es necesario dispositivos que consuman esos datos con un fin, en este caso realizar una acción, de ahí el nombre de actuadores.

Al igual que en el anterior caso, según las necesidades del usuario tendrá los actuadores divididos por secciones, o de forma más genérica. Un ejemplo práctico sería la calefacción, que podría tener una genérica para todo, o un sistema más individualizado

Además el cliente no tiene porqué automatizar todo, puede dejar trabajos manuales como las ventanas o controlar el riego.

#### 4.1.3. Controladores

Estos dispositivos van a ser los nodos centrales por realizar una comparación con el esquema de las empresas. Deben que poder coexistir de manera conjunta varios de ellos en la red sin entorpecerse en sus tareas.

Tienen como objetivo recopilar toda la información del entorno, y en el caso de que se requiera, que puedan activar algunos de los actuadores que el usuario ordene. Por lo tanto tiene que ser capaz de sincronizar la interfaz con el estado de los dispositivos de la red.

Esta información recopilada tiene que ser almacenada, dando la opción que sea de manera local o en un servidor. Además deben poder gestionar todos esos datos y enviarlos si sea necesario.

#### 4.1.4. Interfaces de usuario

Para que el usuario pueda acceder tanto a la información de monitorización como a la gestión de la plataforma, es necesario disponer de una interfaz de usuario.

Respecto a las interfaces a considerar, pueden encontrarse diferentes tipos de escenarios: situada en cada controlador donde una pantalla muestre los últimos datos, o una interfaz web donde se puedan consultar los datos con unas credenciales, accesible a través de internet.

## 4.2. REQUISITOS DE LA PLATAFORMA A DESPLEGAR

Dichas interfaces, a parte de mostrar los datos en forma de tablas, pueden tratarlos para mostrar gráficas, realizarse previsiones futuras o consultar el estado actual de los dispositivos.

Por último, sería necesario que a través de estas se pudiese controlar el estado de los diferentes actuadores.

### **4.2. Requisitos de la plataforma a desplegar**

Tras haber planteado el entorno de trabajo y las diferentes categorías de dispositivos, se podrían agrupar los diferentes requisitos encontrados en el escenario:

- El servicio tiene alta disponibilidad.
- Debe de existir una red IP donde se interconecten los dispositivos
- Los sensores deben tener uno o varios dispositivos de lectura del entorno.
- Los sensores deben publicar datos.
- Los actuadores deben tener uno o varios dispositivos para realizar diferentes acciones.
- Los actuadores deben poder leer los datos de los sensores.
- Los controladores deben leer todos los datos publicados por los nodos.
- Los controladores deben almacenar los datos.
- Deben de poder añadirse nuevos dispositivos a la red.
- Estos datos almacenados deben poder ser visualizados.
- Debe existir una conexión externa a internet para la subida de estos datos.

Tras este desglose de las funcionalidades que debe abarcar el proyecto, se puede proceder realizando el diseño de este para su posterior implementación.

### **4.3. Diseño general del proyecto**

Por tener un ejemplo de un invernadero, podemos observar la imagen 4.1, de la web de Novagric [22]:

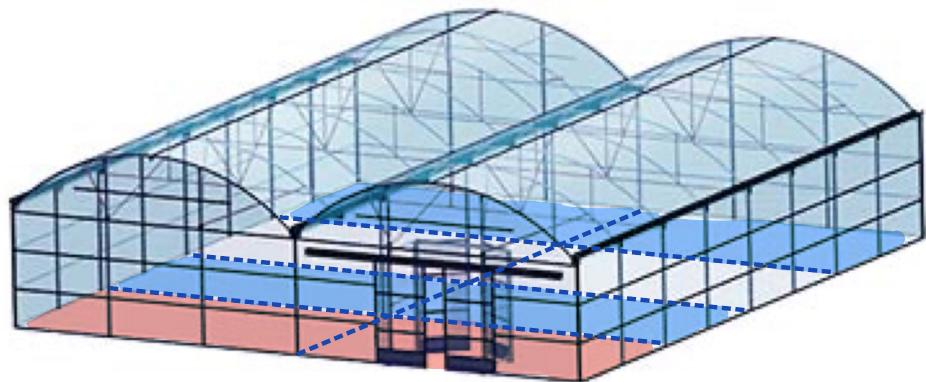


Figura 4.1: Dibujo de un invernadero tipo capilla (Novagric)

En base al dibujo anterior se va a plantear la división del terreno en 8 cuadrantes, mostrado con una vista aérea en el siguiente diagrama 4.2:

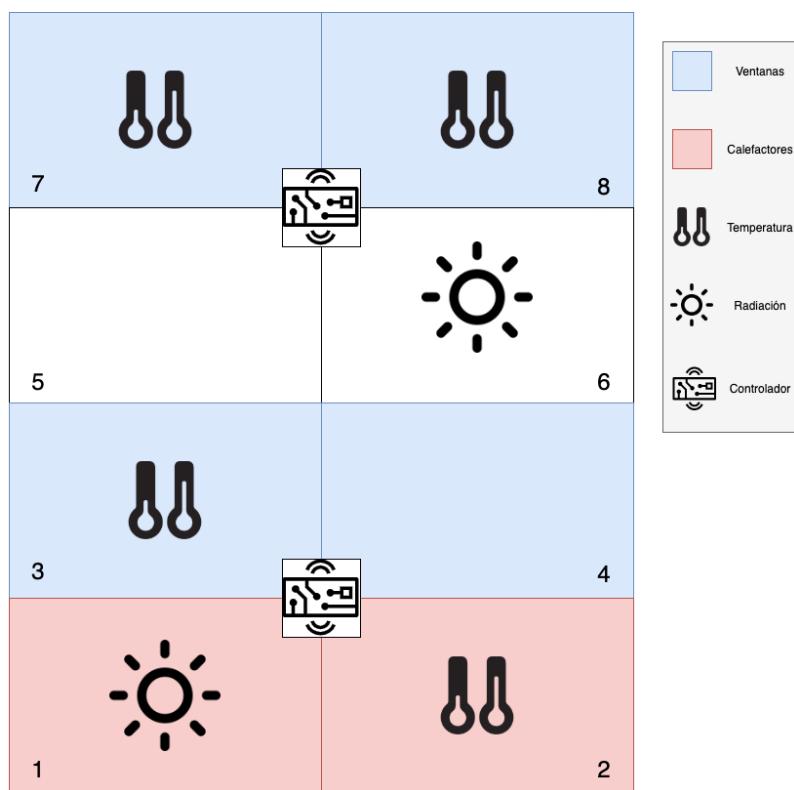


Figura 4.2: Diagrama de un invernadero con vista aérea

Hay una leyenda en la figura indicando que es cada dispositivo, aclarando en la siguiente lista su funcionamiento:

- **Sensor de temperatura.** Sensores situados en las zonas 2, 3, 7 y 8. Publica los datos recopilados.

#### 4.4. DISEÑO DE CLASES DE CADA DISPOSITIVO

- **Sensor de radiación.** Sensor situado en las zonas 1 y 6. Publica los datos recopilados.
- **Ventanas.** Cada recuadro azul indica que hay motores de ventanas en esa zona, como los mostrados en la figura 2.1
- **Calefacción.** Cada recuadro rojizo hace referencia a la zona de actuación de los calefactores. En función de los datos suministrados por el sensor de temperatura 1 se activa o desactiva.
- **Controlador.** Encargado de recoger los distintos datos.

Además de estos dispositivos, hay un gestor de la base de datos para almacenar los datos recogidos por los controladores y una interfaz web. Estos no se han incluido en el escenario porque pueden estar desplegados en otro lugar.

#### 4.4. Diseño de clases de cada dispositivo

Cada nodo mostrado como se ha podido ver durante el análisis dispone de diferentes conexiones y funciones a realizar, las cuales se van a ver reflejadas en la esta sección:

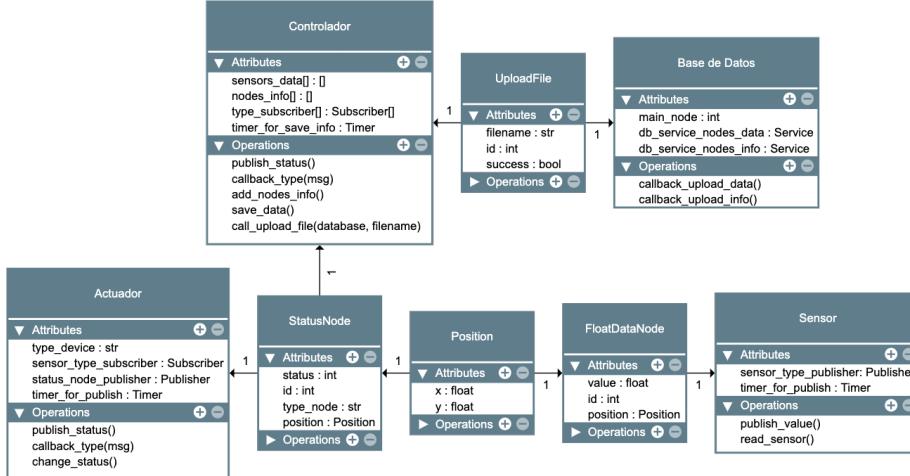


Figura 4.3: Diagrama de clases

En este diagrama están representados los sensores y actuadores de manera genérica. En función del número de datos que recopile un sensor, existirá el mismo número de *publicadores* o *publishers*. Y en el caso de los datos necesarios por un actuador para realizar alguna acción, habrá ese mismo número de suscriptores.

Además en el caso del controlador, en el array mostrado *type\_subscriber*, tendrá uno por cada tópico diferente en la red.

Las clases UploadFile, StatusNode, Position y FloatDataNode son autogeneradas a partir de un archivo por ROS2, por lo tanto están mostradas en la figura 4.4 de la siguiente sección.

## 4.5. Diseño de interfaces en ROS2

En esta sección se va a tratar la parte de comunicación entre los diferentes nodos. Para ello se tienen que definir las estructuras a utilizar para la comunicación, ya que es necesario indicar el tipo de mensaje a utilizar.

Un tópico [31] representa la unidad de información que puede ser producida o consumida por una aplicación DDS. ROS 2 las renombra interfaces debido a que las utiliza para comunicación publicador-suscriptor y cliente-servidor.

En el siguiente diseño se muestran las diferentes estructuras utilizadas.

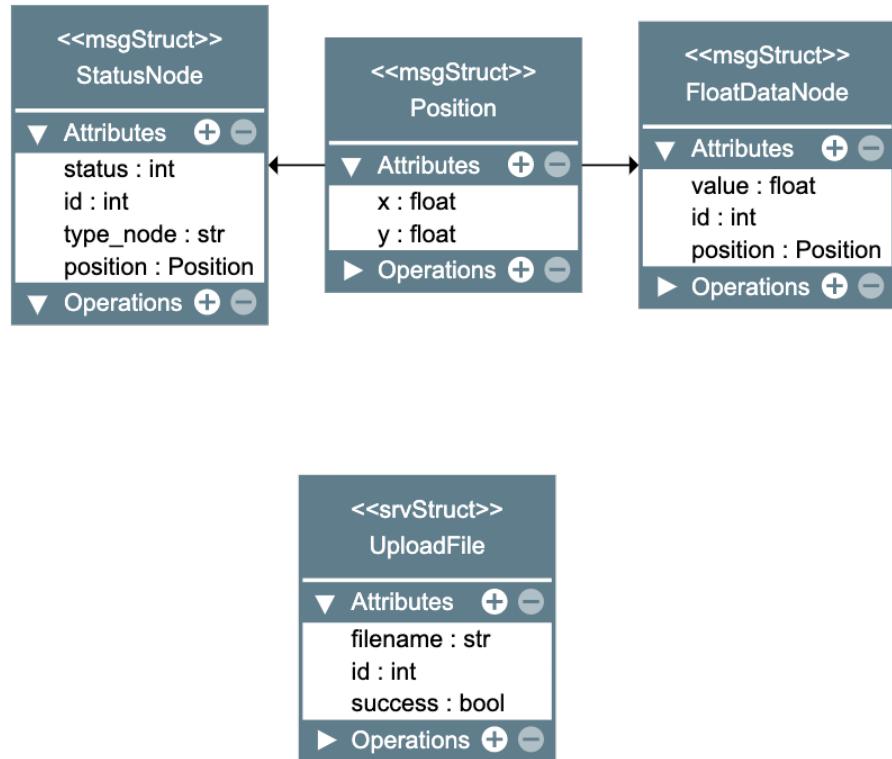


Figura 4.4: Estructuras definidas utilizadas en los tópicos

Estas estructuras en DDS se definen en archivos Interface Description Language (IDL), los cuales definen una estructura estandarizada para todas las arquitecturas contempladas, y a partir de ella son generados los diferentes ficheros necesarios para utilizarlos como tipo de mensaje en un tópico. En ROS2 son conocidas como interfaces, subdivididas en dos grupos, los servicios y los mensajes.

#### 4.5. DISEÑO DE INTERFACES EN ROS2

Estos pueden ser visualizados en las cabeceras de la figura anterior, explicadas a continuación:

- **msgStruct**, hace referencia a un mensaje, utilizado de forma pub-sub en un tópico.
- **srvStruct**, hace referencia a un servicio, utilizado en una conexión cliente-servidor. Esta conexión se utiliza en la comunicación del Controlador y el gestor de la Base de Datos.

Ahora se van a analizar las diferentes interfaces utilizados en este despliegue, los cuales llevan asociado un tipo del conjunto anterior.

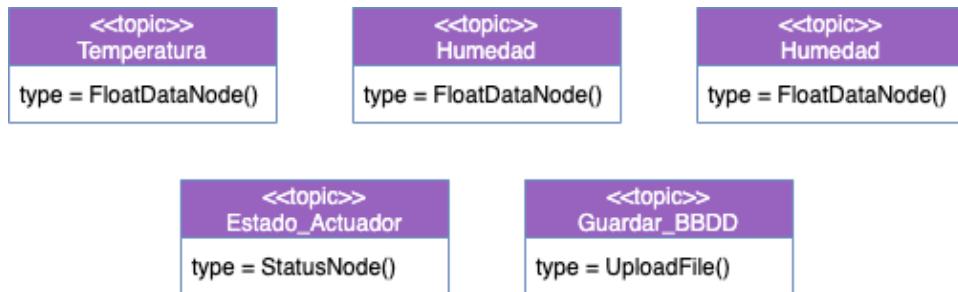


Figura 4.5: Interfaces

Como podemos observar, existen 5 tipos de interfaces, 4 de ellas referentes a los nodos mostrados en el esquema 4.2, y otra referente al almacenamiento de los datos.

Cada uno de estos interfaces tienen asociadas un conjunto de Quality of Service (QoS), utilizados en ROS2 [1]. QoS hace referencia a unas políticas de comunicación que se pueden establecer entre los nodos. En la siguiente lista se van a explicar las más destacables para este proyecto:

- *Keep last*, haciendo referencia al número de datos a almacenar en la cola, ya que si no se podrían almacenar todos.
- *Queue size*, valor utilizado en *keep last* si está asignado
- *Reliable*, para asegurar que los datos lleguen a su destino
- *Volatile*, no se hace intento de hacer persistentes las muestras.

Para finalizar esta sección, se va a mostrar un diagrama de comunicación entre los diferentes nodos, pudiendo observar como operan con los tópicos en la red.

Por aclarar, los diferentes nodos están mostrados con óvalos, y en el caso de los tópicos, se utilizan los rectángulos.

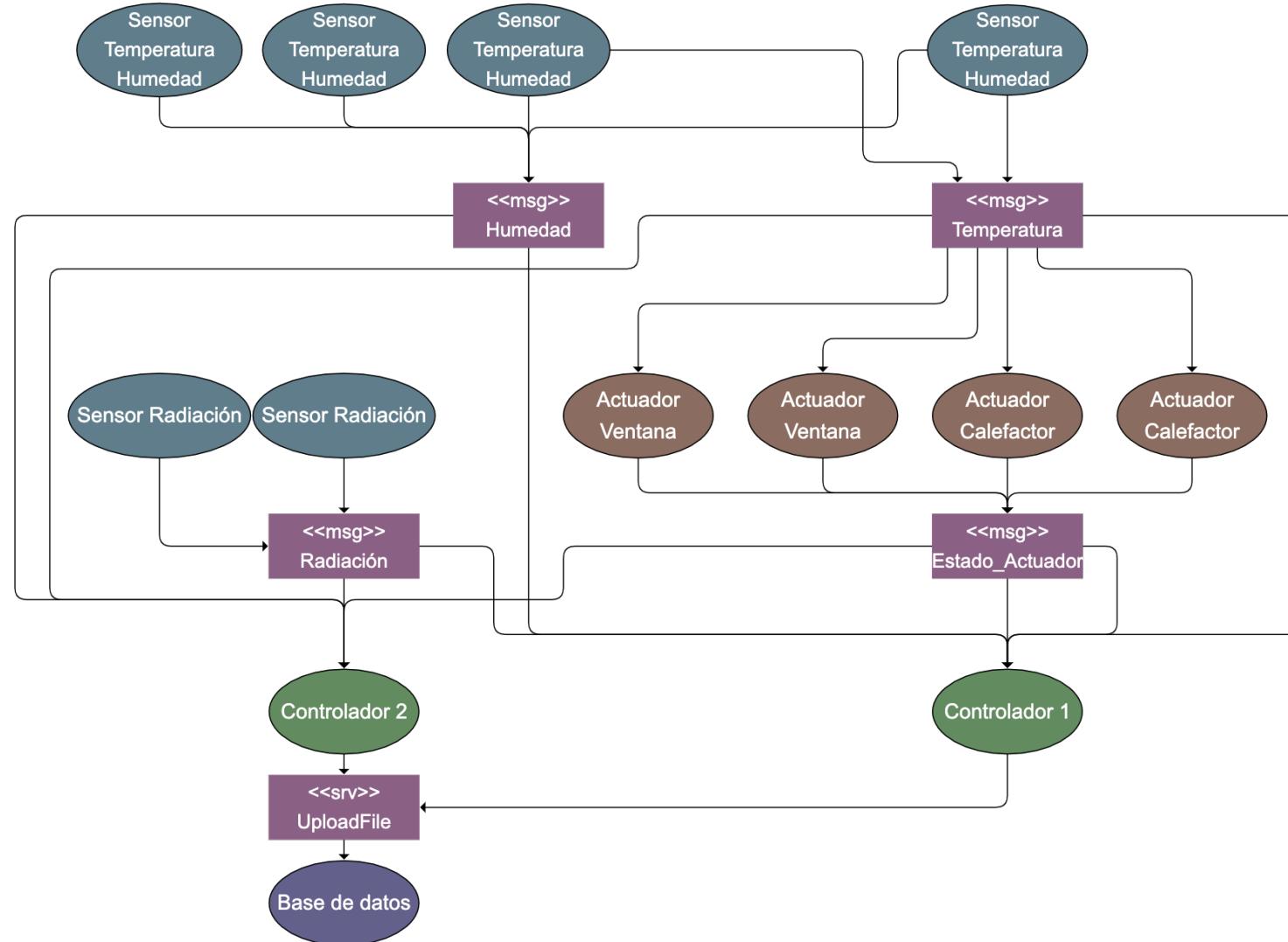


Figura 4.6: Diagrama de comunicación entre nodos

## 4.6. Diseño de la Base de Datos

Tras las explicaciones de los datos tratados en las secciones anteriores, es necesario esquematizar que datos se van a guardar y como van a ser distribuidos. Para ello se van a definir dos tablas principales: sensores y actuadores. Estos deberán constar de los siguientes componentes:

- **Id.** Almacena el id del dispositivo
- **Tipo.** Almacena el tipo de dato producido
- **Posición.** Almacena la posición del nodo

Aquí no se incluye ningún valor registrado sobre el estado del nodo o que valor esta obteniendo de los sensores, si no información más relacionada con su tipo o localización, información que varía poco a lo largo del tiempo.

Únicamente va a existir una entrada por cada Id definido en la tabla. En el caso de que llegue nueva información por si el dispositivo se ha modificado su ubicación, o si transmite nuevos tipos de datos, esta entrada será actualizada.

Por otro lado, se va a tener una tabla por cada tipo de medida (temperatura, humedad...) y tipo de actuador (ventana, calefacción...). La estructura definida para este caso es:

- **Id.** Id del dispositivo
- **Valor.** Valor recogido por el sensor, o estado del actuador
- **Timestamp.** Tiempo haciendo referencia a cuando se ha enviado este dato por la red.

Un ejemplo de tabla de *Temperatura* con algunos datos sería la siguiente:

Id	Valor	Timestamp
11	23	"10/26/21, 11:30:15"
12	21	"10/26/21, 11:30:15"
13	20	"10/26/21, 11:30:15"
11	22	"10/26/21, 11:15:20"
12	21	"10/26/21, 11:15:20"

Figura 4.7: Tabla de Temperatura

En este caso se añade una entrada a la tabla por cada muestra obtenida, para luego poder visualizar los datos a lo largo del tiempo.

#### 4.7. Diseño de la interfaz web

Esta es la parte final del capítulo, ya que tras analizar diseño e implementar, es necesario que el cliente pueda observar todos estos datos. Para ello se ha planteado un entorno web adaptable a distintos dispositivos desde donde se pueda interactuar para consultar información o mandar órdenes al sistema.

Este debe poder acceder a la base de datos mencionada anteriormente, y mostrar los datos en diferentes categorías.

La web debería constar de los siguientes componentes:

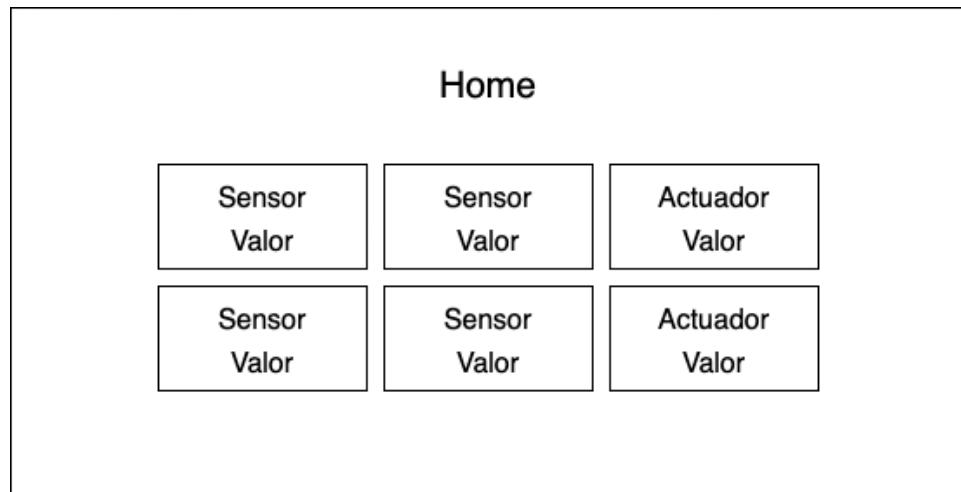


Figura 4.8: Página principal

La página principal será la primera pantalla de la interfaz, donde se muestra en un dashboard el valor o estado actual de los diferentes nodos que hay desplegados en el escenario, pudiendo ver de un vistazo la información.

El diagrama muestra una estructura de interfaz web. En la parte superior central, se encuentra el título "Tabla de datos". A continuación, se dispone de una tabla de datos. La tabla tiene una cabecera con tres columnas: "Id", "Valor" y "Timestamp". Abajo de la cabecera, hay tres filas de datos. Los datos son los siguientes:

Id	Valor	Timestamp
1	20	11:30:15
2	21	11:30:15
3	20	11:30:15

#### 4.7. DISEÑO DE LA INTERFAZ WEB

Figura 4.9: Tablas de datos

La figura anterior 4.9 muestra todos los datos recopilados en la base de datos, subdivididas según el tipo de dato (temperatura, humedad..). Además sería conveniente poder buscar algún valor por hora, filtrar por id o incluso ordenar por parámetros.

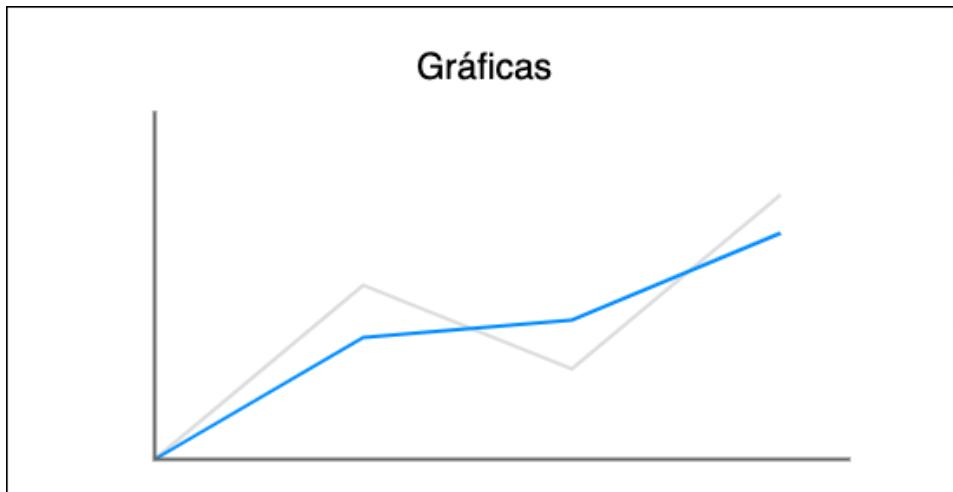


Figura 4.10: Gráficas

Sería idónea la implementación de una sección con los diferentes datos mostrados en gráficas, ya que facilita al usuario la interpretación y la visualización de la información respecto al tiempo, pudiendo comparar los diferentes sensores de una forma más sencilla.

Además es complicado recordar que nodo está situado en cada posición, por lo que un mapa con cada uno situado sería una funcionalidad a considerar. Esto vendría a ser similar a la figura 4.2. Para visualizarlo al igual que los ejemplos anteriores, se adjunta una imagen 4.11 a continuación del diseño planteado:

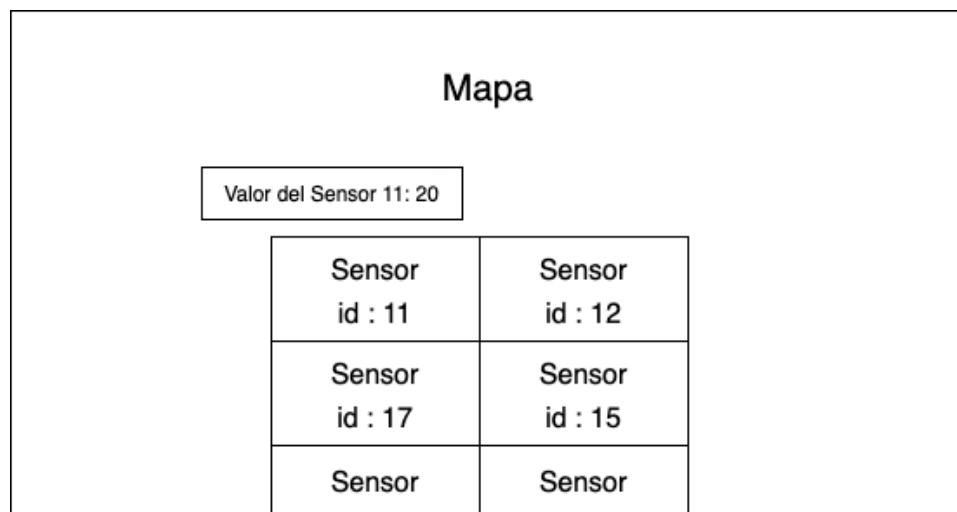


Figura 4.11: Mapa para visualizar la localización de los componentes

Una funcionalidad interesante a implementar sería que seleccionar un sensor, el apartado de *Valor del sensor* se actualice y muestre el estado actual de este.

# Capítulo 5

## Implementación

El proceso de hacer realidad este proyecto y poder tenerlo desplegado ha tenido diferentes pautas a seguir, empezando por la parte de ROS 2, ya que era la esencial en el funcionamiento de nuestra plataforma.

Tras esto se han ido añadiendo las diferentes secciones comentadas en el análisis, haciendo una breve descripción de porqué se ha escogido esa tecnología para cada tarea, además de los diferentes problemas encontrados al realizarlo.

### 5.1. ROS 2

Para la creación del proyecto es necesario tener instalado ROS 2 y todas sus dependencias, explicado en la documentación de su web [2]. En este caso se ha utilizado la versión Foxy.

Además hay que añadir diferentes scripts a nuestra shell para poder ejecutar los programas, o tener disponible el autocompletado de los comandos. Tras la instalación y configuración, se puede ejecutar algún programa de ejemplo para probar si se ha completado todo de manera correcta.

ROS 2 trae diferentes herramientas, donde se van a destacar las más utilizadas en el proyecto:

- **ros2 command.** Utilizado para diferentes tareas como ejecutar programas de ros2, creación de paquetes, consultar información de los nodos en ejecución, consulta de tópicos y más para cosas relacionadas con la parte de depuración.
- **colcon.** Utilizado para compilar el proyecto de ros, pudiendo seleccionar los diferentes paquetes por si está dividido en diferentes secciones.
- **rqt\_graph.** Herramienta gráfica que genera un esquema de la red con los diferentes nodos y tópicos, con sus respectivas conexiones, pudiendo mostrar u ocultar según unos parámetros a seleccionar.

Por lo tanto, para empezar se va a crear el entorno de trabajo, creando una carpeta para ello y tras esto, crear un directorio *src* en su interior. Ahí tras ejecutar colcon build que genera las diferentes carpetas necesarias se podrán crear los diferentes paquetes.

### 5.1.1. Creación de las diferentes interfaces utilizadas en los tópicos

Para poder utilizar los tópicos definidos en el análisis, es necesario crear un paquete donde se van a implementar esas interfaces. Para ello, situado en el directorio *workdir/src* se ejecutaría el comando para crear un paquete, `ros2 pkg create my_tfg_interfaces`.

Tras esto, será generada la carpeta *my\_tfg\_interfaces*, donde se será necesario crear los diferentes directorios y ficheros, para tener una estructura similar a la siguiente:

```

1 src/my_tfg_interfaces/
2   msg/
3     FloatDataNode.msg
4     StatusNode.msg
5     Position.msg
6   srv/
7     UploadFile.srv
8   CMakeLists.txt
9   package.xml

```

Listing 5.1: Estructura del directorio de trabajo

Tras esto, el código de los archivos creados *.msg* y *.srv* es muy simple, por lo que vamos a visualizar *FloatDataNode.msg*:

```

1 float32 data
2 uint16 device_id
3 Position position

```

Listing 5.2: Código *FloatDataNode.msg*

Como se puede observar, solo hay que declarar las variables que se quieran utilizar en el envío. Esto es debido al uso de IDL, que solo hace necesario la definición de estas, ya que ROS genera los diferentes ficheros necesarios. Por aclarar, *Position* en este caso es otro archivo, como si fuese un objeto con las otras variables.

En el caso de los servicios, el formato cambia un poco ya que hay un *request* y un *reply*, por lo tanto se escribe dividido como en el ejemplo a continuación:

```

1 uint16 device_id
2 string filename
3 ---
4 bool success

```

Listing 5.3: Código *UploadFile.srv*

Este código corresponde a `UploadFile.srv`, utilizado en la comunicación del nodo controlador y el nodo gestor de la base de datos.

Tras esto sería necesario modificar los archivos `CMakeList.txt` y `package.xml`, añadiendo las dependencias correspondientes y los archivos creados.

Por último, tras cada nuevo elemento o modificación de cada paquete, es necesario volver a compilar cada paquete.

### 5.1.2. Desarrollo de los nodos

Al igual que se ha creado un paquete en el punto anterior, en este caso se aplicaría el mismo procedimiento, pero el comando no sería exactamente el mismo, ya que sería necesario indicar el tipo de lenguaje a utilizar.

Como lenguaje se va a utilizar Python, aunque sería posible utilizar otros lenguajes, como C++. Por lo tanto para generar el paquete ejecutamos:

```
1 ros2 pkg create my_tfg_pkg --build-type ament_python \
2 --dependencies rclpy
```

Listing 5.4: Comando para la creación de un paquete de Python con ROS 2

Esto es para la creación del paquete especificando el tipo de lenguaje y las dependencias, en este caso `rclpy` que contiene todo lo necesario para poder desarrollar los nodos.

Tras todo esto, el directorio tendría la siguiente estructura:

```
1 src/my_tfg_pkg/
2     my_tfg_pkg/
3         __init__.py
4         test/
5         resource/
6         setup.cfg
7         setup.py
8         package.xml
```

Listing 5.5: Estructura del directorio de desarrollo con Python

Para poder construir los diferentes nodos, tenemos que añadir los respectivos ficheros en la carpeta `src/my_tfg_pkg/my_tfg_pkg`.

Antes de empezar con la implementación, según la comunidad, la estructura estándar de los nodos tiene que ser la siguiente:

```

1 #!/usr/bin/env python3
2 import rclpy
3 from rclpy.node import Node
4
5
6 class MyCustomNode(Node): # MODIFY NAME
7     def __init__(self):
8         super().__init__("node_name") # MODIFY NAME
9
10
11 def main(args=None):
12     rclpy.init(args=args)
13     node = MyCustomNode() # MODIFY NAME
14     rclpy.spin(node)
15     rclpy.shutdown()
16
17
18 if __name__ == "__main__":
19     main()

```

Listing 5.6: Plantilla genérica para la creación de nodos

Se puede observar en el código un método *main* donde se crea un objeto de nuestra clase, y con el método *spin*, se mantiene en ejecución ese nodo. Con esta base ya se pueden empezar a crear los nodos necesarios.

Por tanto, se va a repasar los diferentes nodos de manera genérica:

- **Sensores.** Productores de información.
- **Actuadores.** Consumidores de esos datos, y productores de otros.
- **Controladores.** Encargados de recopilar todos los datos.

En las figuras mostradas durante el análisis, podemos observar el diseño definido de estos archivos concretamente en la 4.3.

#### 5.1.2.1. Sensores

Van a realizar una lectura del sensor cada cierto tiempo, llamando al método *read\_sensor()*. Esta función será la principal diferencia entre este tipo de nodo, ya que para su implementación será necesario una librería u otra. Tras esta lectura, estos datos serán publicados en la red. Para ello es necesario tener:

- Un **publisher**, que defina el tópico donde se va a publicar
- Un **timer**, que establezca el tiempo cada vez que se va a realizar esa publicación. Este tendrá como parámetro la función de *publish\_value()*, que será llamada en el periodo definido.

Su declaración viene explicada en la documentación [2], pero por abreviar la búsqueda, esta se realiza de la siguiente forma:

```

1 self.temperature_publisher_ =
2     self.create_publisher(FloatDataNode, "temperature", 10)
3 self.sensor_timer_ =
4     self.create_timer(20.0, self.publish_temperature_data)

```

Listing 5.7: Publisher del sensor de temperatura

Se va a definir el publisher, ya que los argumentos pueden ser confusos:

- **FloatDataNode**, hace referencia al tipo de dato publicado en el tópico
- **"temperature"**, es el nombre del tópico donde se van a publicar los datos
- **10**, hace referencia al número de mensajes a conservar en el historial. También se puede declarar un elemento QoSProfile donde declaremos las diferentes QoS [1] a aplicar.

El timer recibe como parámetros el tiempo y la función que va a llamar, donde el publisher publica el mensaje.

Según el los diferentes tipos de sensores que tenga conectados este nodo, le será necesario ese mismo número de publishers con sus respectivas funciones.

### 5.1.2.2. Actuadores

A diferencia de los sensores, estos van a tener un publisher para el estado de este nodo, el timer para la publicación de esta información y un subscriber.

Este subscriber es el nuevo añadido a diferencia del anterior nodo. Para ver su estructura vamos a visualizar la siguiente declaración:

```

1 self.temperature_subscriber_ = self.create_subscription(
2     FloatDataNode, "temperature", self.callback_sensor_data, 10)

```

Listing 5.8: Subscriber del actuador

Los parámetros hacen referencia al tópico del que se quieren leer los datos. El método `callback_sensor_data` maneja los datos recibidos, comparando la temperatura en este caso con un posible valor de acción. Como se comenta en la parte de análisis, un ejemplo es cuando la temperatura baje de 12°C, la calefacción se encienda y tras esto, se publique el nuevo estado.

### 5.1.2.3. Controladores

El nodo **controlador** posee más funcionalidades que los anteriores, ya que está orientado a utilizarse en un dispositivo central con más potencia.

Posee un subscriber por cada tipo de tópico en la red, y tras la recepción de cada dato es guardado en un diccionario de Python. Este diccionario tras un tiempo definido con un timer almacena los datos en un json, para posteriormente contactar con el nodo gestor de la base de datos. Esto lo hace en los dos métodos siguientes

```

1 def call_upload_file(self, json_name):
2     client = self.create_client(UploadFile, "upload_db")
3     while not client.wait_for_service(1.0):
4         self.get_logger().warn("Waiting for DB Service...")
5
6     request = UploadFile.Request()
7     request.device_id = self.status_controller_.device_id
8     request.filename = json_name
9
10    future = client.call_async(request)
11    future.add_done_callback(
12        partial(self.callback_upload_file,
13                device_id=self.status_controller_.device_id,
14                filename=json_name))
15
16 def callback_upload_file(self, future, device_id, filename):
17     try:
18         response = future.result()
19         if response.success:
20             self.get_logger().info("Upload to Database done.")
21     except Exception as e:
22         self.get_logger().error("Service failed %r" % (e,))

```

Listing 5.9: Creación de cliente de ROS2 en Python

Aquí se está gestionando una comunicación cliente-servidor, debido a que pub-sub no es la más indicada en este caso, ya que es necesario la obtención de respuesta por parte del gestor de la base de datos.

Se pueden visualizar métodos asíncronos ya que no se puede detener la ejecución del controlador de almacenar los demás datos que llegan en ese tiempo.

Por estructurar el código mostrado, primeramente se crea el cliente según el tópico a utilizar y se espera una conexión por parte del servidor. Tras esto se crea el objeto request que va a ser enviado. Por último se realiza la espera asíncrona de la response, y según su valor, se muestra en el gestor de logs que ha sido subido o se lanza una excepción.

#### 5.1.2.4. Ejecución y herramientas utilizadas

Terminando esta sección se van a mostrar en las siguientes capturas la ejecución de los nodos, diferentes herramientas en la shell con el comando ros2, y por último el estado de la red mostrado con rqt\_graph.

## 5.1. ROS 2

```
pparrilla@Asus:~$ ros2 run my_tfg_pkg temp_hum_publisher --ros-args -r __node:=temperature_humidity -p device_id:=14 -p temperature:=19.0 -p humidity:=23.0
[INFO] [1636487210.654539199] [temperature_humidity]: Temp_Hum_Sensor_14 has been started

pparrilla@Asus:~$ ros2 run my_tfg_pkg irradiance_publisher --ros-args -r __node:=irradiance -p device_id:=12
[INFO] [1636487216.447639421] [irradiance]: irradiance_12 has been started

pparrilla@Asus:~$ ros2 run my_tfg_pkg heater_node --ros-args -p device_id:=30
[INFO] [1636487220.370096802] [heater_32]: Heater_30 has been started.

pparrilla@Asus:~$ ros2 run my_tfg_pkg controller_node --ros-args -p device_id:=1
[INFO] [1636487223.997637247] [controller_1]: Controller_1 has been started.

pparrilla@Asus:~$ ros2 run my_tfg_pkg firebase_service
[INFO] [1636487150.991655432] [firebase_service]: Firebase service has been started.
```

Figura 5.1: Ejecución de los diferentes nodos con parámetros

Estos parámetros hacen referencia al id del dispositivo o al valor de temperatura que va a estar emitiendo ese nodo, ya que actualmente son virtuales y no leen de ningún sensor físico.

```
pparrilla@Asus:~$ ros2 node list
/controller_1
.firebaseio_service
/heater_32
/irradiance
/temperature_humidity
pparrilla@Asus:~$ ros2 topic list
/humidity
/irradiance
/parameter_events
/rosout
/status_actuator
/temperature
pparrilla@Asus:~$ ros2 topic echo /temperature
data: 19.0
device_id: 14
position:
  x: 0.0
  y: 0.0
---
pparrilla@Asus:~$ pparrilla@Asus:~$ ros2 topic list
/humidity
/irradiance
/parameter_events
/rosout
/status_actuator
/temperature
pparrilla@Asus:~$ ros2 topic echo /status_actuator
work_status: 0
device_id: 30
device_type: heater
position:
  x: 0.0
  y: 0.0
---
pparrilla@Asus:~$
```

Figura 5.2: Herramientas con el comando ros2

Se pueden observar distintos comandos de ros2 en la anterior figura:

- **ros2 node list** muestra la lista de nodos actualmente ejecutándose en la red
- **ros2 topic list** muestra la lista de tópicos en la red
- **ros2 topic echo /temperature** muestra por pantalla los datos publicados referentes a ese tópico de temperatura

Hay más comandos similares, como **ros2 interfaces show interface.msg** que mostraría las variables que implementa esa interfaz, muy útil cuando se utilizan interfaces no propias. Pero los más utilizados son los mostrados en la figura 5.2.

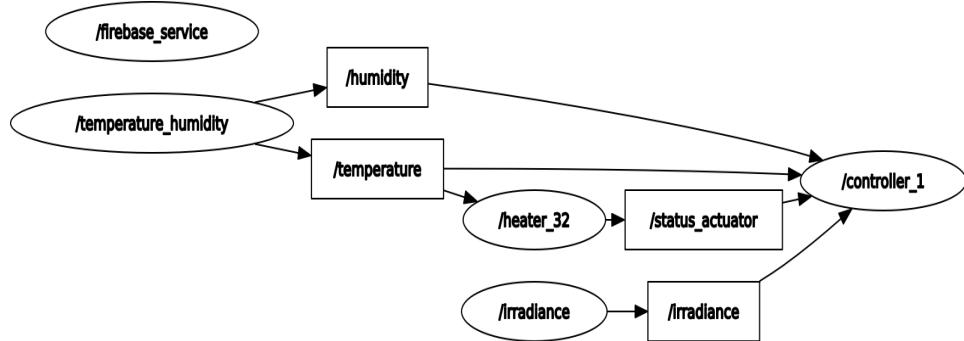


Figura 5.3: Gráfico generado con rqt-graph de la red

Por aclarar al igual que en la figura 4.6 los nodos están representados con el óvalo, y los rectángulos son los tópicos.

Esta herramienta, como se ha comentado, genera un gráfico de los distintos nodos y tópicos ejecutándose en la red, mostrando la comunicación existente. Los tópicos utilizados en conexiones cliente-servidor no los añade a la red, por eso controller\_1 y firebase\_service no aparecen conectados.

### 5.1.3. Micro ROS en microcontroladores

Además de la utilización de ROS 2 en este proyecto, se han querido utilizar microcontroladores ya que son utilizados en casi cualquier producto robótico, estas tienen una baja latencia en cuestión a trabajo a tiempo real y consumen menos energía, además de su precio que suele ser más económico.

Para ello, es necesario la utilización de micro-ROS [20], que es la adaptación de la tecnología a estos microcontroladores. Además ha sido necesario utilizar FreeRTOS [15]. Esto es un sistema operativo a tiempo real donde encima se va a ejecutar micro-ROS.

Una vez preparado el entorno como viene explicado en la documentación de micro-ROS, es necesario crear los siguientes ficheros:

- **app.c**, que contiene la aplicación que se va a ejecutar
- **app-colcon.meta**, que contiene la información específica necesaria para la compilación con *colcon*.

Como se puede observar, ahora hay que utilizar C como lenguaje, donde no existen las clases, por tanto la estructura del fichero va a variar, pero no mucho ya que se realizará la mayoría de cosas en el propio main, llamando a diferentes métodos.

El objetivo va a ser utilizando una placa ESP32 leer de un sensor DHT-11 la temperatura y humedad, por lo tanto es necesario también incluir en el proyecto la librería específica para realizar la lectura de datos.

La creación de un publisher y un timer en este caso ha sido la siguiente:

```

1 // create publisher for temperature
2 RCCHECK(rclc_publisher_init_default(
3     &temperature_publisher,
4     &node,
5     ROSIDL_GET_MSG_TYPE_SUPPORT(custom_node_message, msg,
6                                     FloatDataNode),
7     "temperature"));
8
9 // create timer
10 rcl_timer_t timer;
11 const unsigned int timer_timeout = 30000;
12 RCCHECK(rclc_timer_init_default(
13     &timer,
14     &support,
15     RCL_MS_TO_NS(timer_timeout),
16     timer_callback));
17
18 // create executor
19 rclc_executor_t executor;
20 RCCHECK(rclc_executor_init(&executor,
21                           &support.context, 1, &allocator));
22 RCCHECK(rclc_executor_add_timer(&executor, &timer));

```

Listing 5.10: Publisher Timer y Executor en C para microcontrolador

Tras esto, cada 30 segundos se ejecuta la función `timer_callback` que se encarga de leer los datos del sensor y publicarlos.

Ya con la aplicación operativa y todas las dependencias añadidas, sería momento de construir el proyecto y flashearlo en el microcontrolador.

El montaje del sensor y la placa tendría la siguiente estructura:

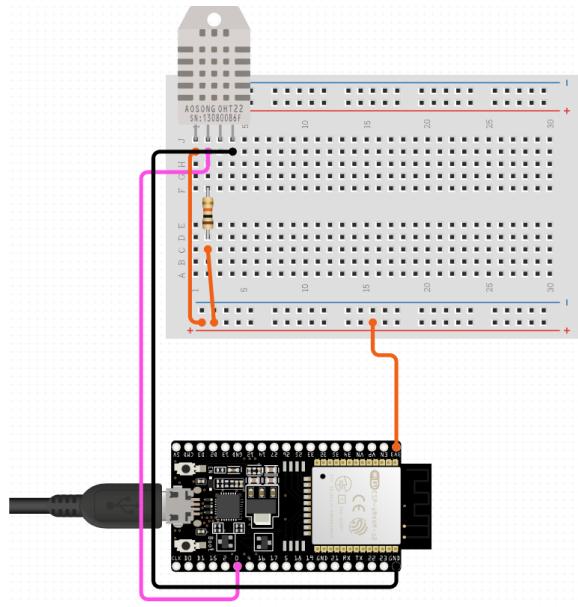


Figura 5.4: Montaje ESP32 con DHT11

Hay que tener en cuenta, que micro-ROS necesita la existencia de un agente para que el microcontrolador pueda contactar con el resto del proyecto de ROS 2. Este puede ser ejecutado en un contenedor de Docker [8] con el siguiente comando:

```
1 docker run -it --rm --net=host microros/micro-ros-agent:foxy \
2     udp4 --port 8888 -v6
```

Listing 5.11: Ejecución Agente de micro-ROS en un contenedor Docker

Este contenedor debería estar ejecutando en uno de los controladores distribuidos por el escenario, o en otro dispositivo dedicado para ello.

#### 5.1.4. Conexión a la base de datos

Como se ha comentado, cada controlador almacena la información en un fichero tipo .json, y tras esto contacta con el gestor de la base de datos mediante cliente-servidor. Para ello tiene que enviar su id y el nombre el fichero.

Por lo tanto, el nodo de conexión a la base de datos debe tener las librerías para poder conectarse a esta.

En este caso se han implementado 2 nodos diferentes:

- **Nodo Firebase.** Este realiza una conexión con Firebase [14], donde el archivo será estructurado en colecciones y documentos según define esta tecnología. El motivo a utilizar esta base de datos es el alojamiento en la nube, ya que facilita el acceso a los datos mediante las distintas plataformas. La librería para poder realizar la conexión es *firebase\_admin*.

- **Nodo SQLite3.** Este crea una conexión con SQLite3 [28] donde se almacenarán los datos de manera local del dispositivo en formato de tablas. Este puede estar ejecutado en cada dispositivo donde se ejecuta el nodo controlador, pudiendo tener la información replicada. La librería para poder realizar la conexión es *sqlite3*.

En cada nodo es necesario presentar una estructura de creación del servidor, método para gestionar la llamada y otro método para subir la información a las respectivas bases de datos.

La explicación para el uso de dos bases de datos distintas...

#### 5.1.5. Simulación de un entorno con diferentes nodos

Tras haber implementado los diferentes nodos, en caso de realizar una simulación del escenario descrito, no es viable la opción de ejecutar cada nodo de manera individual en una terminal como se muestra en la figura 5.1, ya que en este caso se quieren lanzar unos 15-20 nodos aproximadamente.

Para estas ocasiones, ROS 2 dota de un tipo de archivo llamado *Launch File*. En este se pueden incluir los diferentes nodos que se quieren ejecutar, indicando los parámetros como el nombre, el valor a producir, la posición y los demás que se hayan asignado durante el desarrollo de estos.

Por lo tanto da la posibilidad de generar un escenario y ejecutarlo desde un fichero, con el comando *ros2 launch*.

Para su uso sería necesario la creación de otro paquete al igual que en los anteriores casos, y tras esos en un archivo tipo *name.launch.py* escribir el código necesario, importando la librería *launch*.

Tras el desarrollo y la ejecución de este fichero, con *rqt\_graph* podemos observar la red de la siguiente forma:

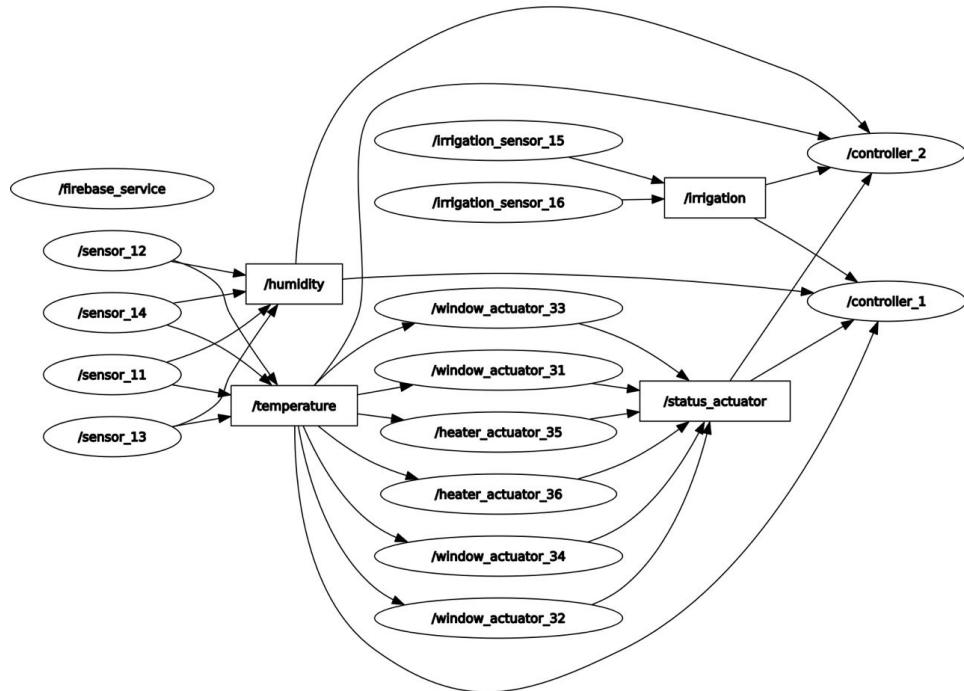


Figura 5.5: Simulación del escenario en la red

Al igual que en el anterior caso 5.3, los óvalos son los nodos y los rectángulos los tópicos.

Este escenario ha sido desplegado utilizando los siguientes dispositivos:

- **2 placas ESP32**, siendo los sensores 11 y 12, leyendo la información de los sensores DHT-11.
- **Raspberry Pi3B**, ejecutando uno de los controladores y almacenando los datos.
- **Portátil**, donde se ha desplegado los demás nodos con el *launch file*.

## 5.2. API REST para obtención de datos

El desarrollo de esta API de comunicación estilo REpresentational State Transfer (REST), ha sido con el objetivo para poder tratar los datos y mostrarlos en la interfaz web explicada en la siguiente sección. Esta tecnología sirve para realizar de intermediario entre la base de datos y interfaz web.

Se van a destacar una serie de ventajas al usar API REST:

- Hay una separación entre el cliente y el servidor, en este caso los usuarios que utilicen la interfaz y el servidor que contiene la base de datos.

- Hace más flexible el entorno de trabajo, ya que esta es independiente al tipo de lenguaje utilizado
- El manejo de errores respecto al acceso es gestionado por estas

Se puede destacar que en el caso de que se cambie de base de datos, ya que se han implementado dos, solo haya que modificar esta API y no sea necesario hacer ningún cambio en esa interfaz. Además, si se quiere utilizar los datos con fines relacionados con la inteligencia artificial, es más viable la conexión para poder analizar estos datos y aplicar algoritmos de entrenamiento o cálculos estadísticos.

Esta ha sido implementada en Python y con el framework FastAPI [12]. Esta tecnología ha sido utilizada para el desarrollo debido a que el despliegue es muy rápido, y en el caso de querer dockerizar este apartado y poder ejecutarlo en diferentes entornos es bastante simple.

Respecto al funcionamiento, este ejecuta Uvicorn [30] que es un servidor ligero Asynchronous Server Gateway Interface (ASGI) para que este continuamente operativo este servicio.

### 5.3. Interfaz Web

Respecto a la interfaz a establecer, se ha seleccionado Angular [5] debido a la flexibilidad, robustez y la disponibilidad del uso de plantillas en comparación a otras alternativas como React o Vue. Además con planteamiento futuro, se podría implementar junto al framework de Ionic para creación de aplicaciones híbridas, pudiendo crear al finalizar una app para los sistemas operativos móviles como IOS y Android.

Angular utiliza *Components*, que son encapsulaciones de código HTML, CCS y Javascript, dividiendo las diferentes estructuras comentadas en la figura 4.8 para un mejor diseño.

En Angular se ha utilizado Material [19] para el diseño de los diferentes elementos de la web. Material es un conjunto de componentes dotando de diseño y diferentes funcionalidades para hacer más cómodo y rápido esta parte con Angular. Gracias a que la documentación es muy completa y posee multitud de ejemplos, se puede adoptar en el proyecto para las diferentes secciones planteadas.

Para la conexión con la API, es necesario crear un *service* en angular, que genere un *Observable* para recibir la información de manera asíncrona. Este será utilizado en los componentes necesarios, pidiendo los diferentes tipos de datos.

Los componentes finalmente implementados han sido los siguientes:

#### 5.3.1. Home

Este componente de la figura 5.6 muestra cada nodo de la red desplegada con la última información recopilada por este. Es la primera página que se va a

visualizar a la plataforma, por lo que tiene que tener la parte esencial de estos datos.

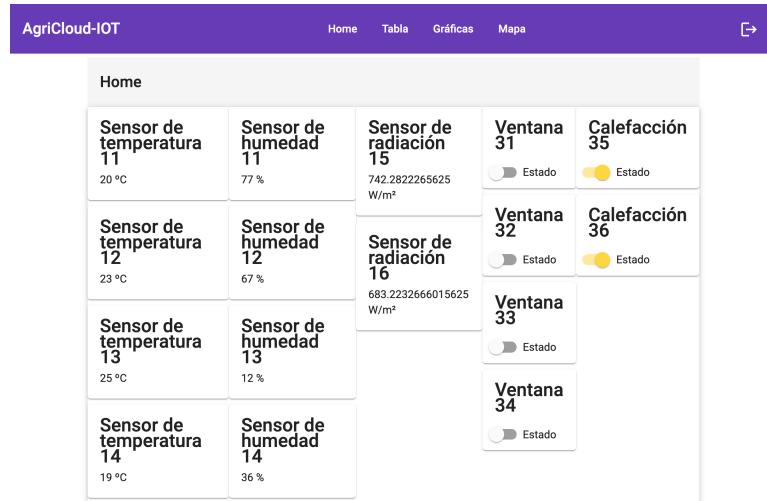


Figura 5.6: Página principal

### 5.3.2. Tabla de datos

Respecto a la tabla de datos, esta ha implementado paginación para una visualización más cómoda, posibilidad de ordenamiento de los datos según cada cabecera o filtrado de estos datos.

Por lo tanto, en el caso de se que quiera buscar por una hora, se escribiría en el filtro la hora, o si se quiere ordenar de mayor a menor tomando como referencia el tiempo, solo sería necesario hacer click en esa columna.

### 5.3. INTERFAZ WEB

The screenshot shows the AgriCloud-IOT web interface. At the top, there is a purple header bar with the text "AgriCloud-IOT" on the left and navigation links "Home", "Tabla", "Gráficas", and "Mapa" on the right. To the far right of the header is a small "Logout" icon. Below the header is a navigation menu with tabs: "Temperatura" (selected), "Humedad", "Radiación", "Ventanas", and "Calefacción". Underneath the menu is a "Filter" section. The main content area displays a table with three columns: "ID Dispositivo", "Valor", and "Tiempo". The table contains five rows of data. At the bottom of the table is a pagination control with "Items per page: 5" and "1 - 5 of 20" along with navigation arrows.

ID Dispositivo	Valor	Tiempo
11	20	10/31/21, 18:19:33
14	19	10/31/21, 18:19:33
13	25	10/31/21, 18:19:33
12	23	10/31/21, 18:19:33
13	25	10/31/21, 18:24:33

Figura 5.7: Tablas de datos

#### 5.3.3. Gráficos

Esta sección hace uso de Google Charts [16], en concreto un paquete disponible para Angular publicado en Github.

Se han escogido las gráficas tipo lineal ya que permiten de un vistazo ver los cambios en los datos producidos en los diferentes sensores.

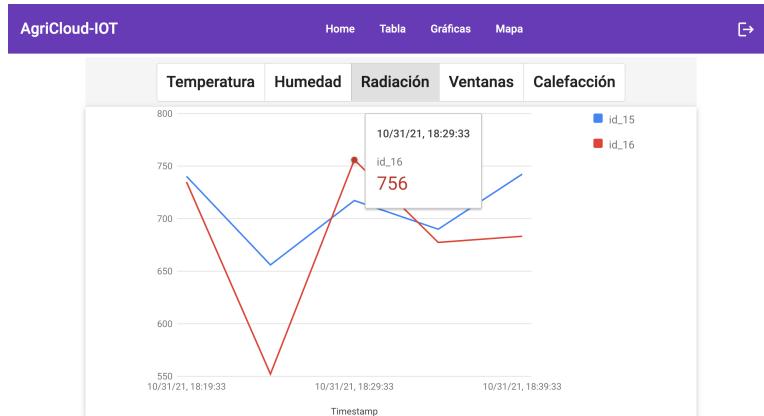


Figura 5.8: Gráficas de radiación

#### 5.3.4. Mapa

En la imagen 5.9 se muestra el mapa final, similar al mostrado en la figura del capítulo de análisis 4.11, para que el usuario que acceda pueda visualizar la posición de cada nodo, además de poder obtener los datos de este al seleccionarlo, mostrados en el recuadro superior.

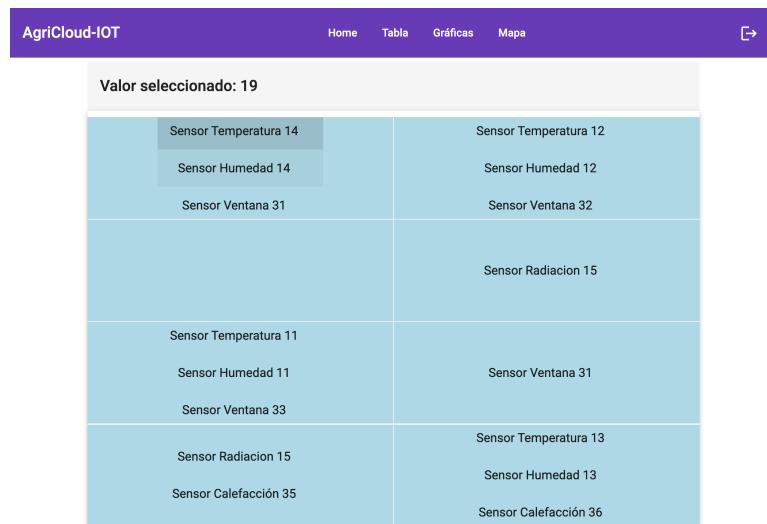


Figura 5.9: Mapa interactivo con los diferentes sensores

### 5.3.5. Login

Se ha querido dejar hecho una página de inicio de sesión, ya que a lo largo de la implementación de la interfaz web, se ha planteado el acceso de diferentes usuarios a esta plataforma.

Por ello se muestra en la siguiente captura 5.10 la introducción de las credenciales de usuario.



Figura 5.10: Login implementado

Esta podría ser la página principal de nuestra interfaz, y tras introducir nuestras credenciales ya se muestre el Home correspondiente a ese usuario.



# Capítulo 6

## Evaluación

En este capítulo se analizará el escenario implementado con los distintos dispositivos explicados a lo largo de la documentación. El objetivo es ver si con una Raspberry Pi se puede realizar este escenario o a mayor escala.

Por recordar, los elementos utilizados son los siguientes:

- Portátil Asus ROG GL752VW con un i7-6700HQ y 24GB de memoria RAM
- Raspberry Pi modelo 3B con 1GB de RAM
- 2 Microcontroladores ESP32

En la implementación, en la sección sobre el fichero de ejecución de múltiples nodos, este es ejecutado en el portátil mencionado para las diferentes pruebas.

En este caso se va a realizar otro archivo diferente, que agrupe los nodos de *Controlador*, *Base de Datos* y los contenedores que actúan como agentes de microROS para recopilar y publicar la información en el dominio utilizado.

El objetivo es ver el consumo de memoria por parte de los procesos anteriores ejecutados en la Raspberry Pi.

Para ver si se producen cambios en la memoria si hay más o menos nodos en la red, estos serán ejecutados en el portátil mencionado.

### 6.1. Escenario 1

Se va a ejecutar un único fichero, el nodo Controlador en la Raspberry Pi, para ello se verá la memoria antes y después de la ejecución. El comando a utilizar va a ser *free -h* que nos devuelve el uso de esta dividida en las diferentes cabeceras.

```
ubuntu@raspberry:~$ free -h
total        used        free      shared  buff/cache   available
Mem:       908Mi       192Mi      355Mi      3.0Mi       360Mi      691Mi
Swap:          0B          0B          0B
ubuntu@raspberry:~$ free -h
total        used        free      shared  buff/cache   available
Mem:       908Mi       223Mi      310Mi      4.0Mi       373Mi      658Mi
Swap:          0B          0B          0B
ubuntu@raspberry:~$
```

```
ubuntu@raspberry:~$ ros2 run my_tfg_pkg controller_node --ros-args -p device_id:=3
[INFO] [1636824260.441687017] [controller_1]: Controller_3 has been started.
```

Figura 6.1: Escenario 1

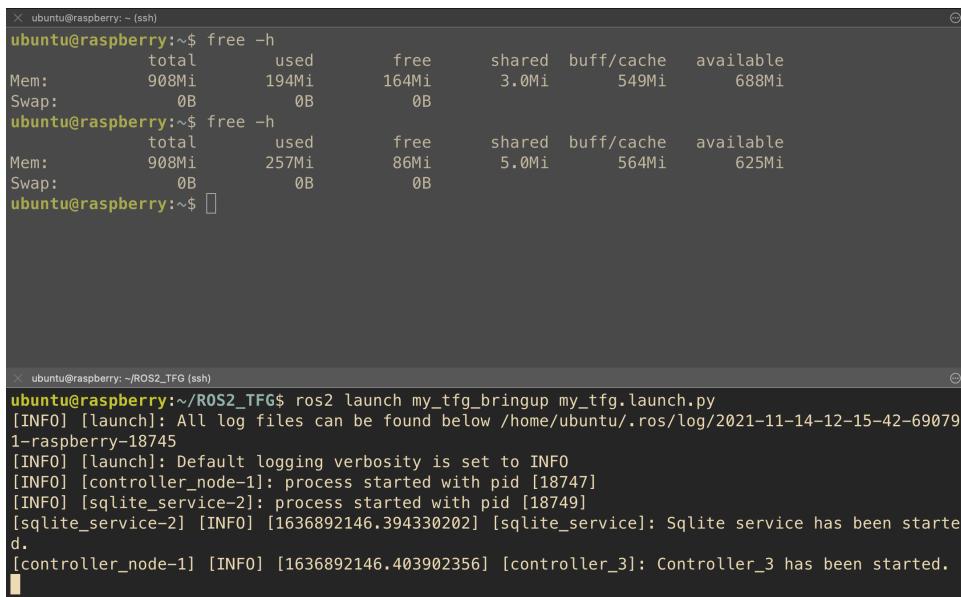
En la figura 6.1 se puede observar dos ejecuciones para medir la memoria en la parte superior, una ejecutada antes y otra después de la llamada al nodo controlador. Por tanto el consumo de memoria producido en este caso es de 41MB.

Tras la ejecución de este escenario unas 5 veces, el consumo de memoria se ha mantenido igual, variando 1MB en algunos casos, pudiendo ser debido a otro proceso del sistema.

## 6.2. Escenario 2

En este caso se han lanzado diferentes nodos en la Raspberry Pi, el Controlador y el gestor de la base de datos. Al igual que en el caso anterior, se va a medir la memoria con *free -h* en el mismo formato:

### 6.3. ESCENARIO 3



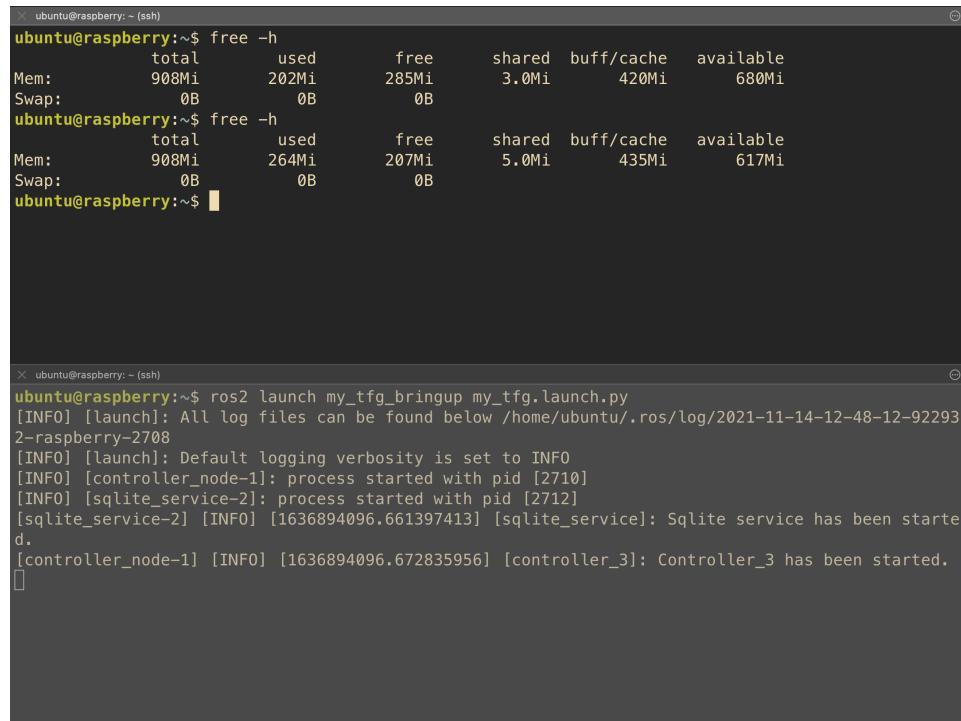
The figure consists of two vertically stacked terminal windows. The top window shows the command `free -h` being run twice. The first run shows total memory of 908MiB, with 194MiB used, 164MiB free, 3.0MiB shared, 549MiB in buff/cache, and 688MiB available. The second run shows total memory of 908MiB, with 257MiB used, 86MiB free, 5.0MiB shared, 564MiB in buff/cache, and 625MiB available. The bottom window shows the command `ros2 launch my_tfg Bringup my_tfg.launch.py` being run. The log output includes: [INFO] [launch]: All log files can be found below /home/ubuntu/.ros/log/2021-11-14-12-15-42-690791-raspberry-18745; [INFO] [launch]: Default logging verbosity is set to INFO; [INFO] [controller\_node-1]: process started with pid [18747]; [INFO] [sqlite\_service-2]: process started with pid [18749]; [INFO] [sqlite\_service-2]: [INFO] [1636892146.394330202] [sqlite\_service]: Sqlite service has been started.; [controller\_node-1] [INFO] [1636892146.403902356] [controller\_3]: Controller\_3 has been started.

Figura 6.2: Escenario 2

Como se puede observar, el consumo de la memoria ha aumentado respecto al anterior ya que se están ejecutando actualmente dos nodos, siendo este de 63MB al ser ejecutado.

### 6.3. Escenario 3

Por último, se van a lanzar todos los nodos mostrados en la figura 5.3 en el capítulo de implementación. Además de esos, estará la Raspberry Pi 3B con los nodos en ejecución, para ver si se refleja toda la comunicación en la memoria. Esta parte será la mostrada en la terminal, ya que es la memoria que se está analizando.



```
ubuntu@raspberry:~ (ssh)
ubuntu@raspberry:~$ free -h
total        used        free      shared  buff/cache   available
Mem:       908Mi       202Mi      285Mi      3.0Mi      420Mi      680Mi
Swap:          0B          0B          0B
ubuntu@raspberry:~$ free -h
total        used        free      shared  buff/cache   available
Mem:       908Mi       264Mi      207Mi      5.0Mi      435Mi      617Mi
Swap:          0B          0B          0B
ubuntu@raspberry:~$ 
```

```
ubuntu@raspberry:~ (ssh)
ubuntu@raspberry:~$ ros2 launch my_tfg Bringup my_tfg.launch.py
[INFO] [launch]: All log files can be found below /home/ubuntu/.ros/log/2021-11-14-12-48-12-92293
2-raspberry-2708
[INFO] [launch]: Default logging verbosity is set to INFO
[INFO] [controller_node-1]: process started with pid [2710]
[INFO] [sqlite_service-2]: process started with pid [2712]
[sqlite_service-2] [INFO] [1636894096.661397413] [sqlite_service]: Sqlite service has been starte
d.
[controller_node-1] [INFO] [1636894096.672835956] [controller_3]: Controller_3 has been started.

```

Figura 6.3: Escenario 3

Al igual que en el escenario 2, el consumo de memoria mostrado en la figura 6.3 es de 63MB.

Por comprobar esta variación, se ha ejecutado este escenario 5 veces, obteniendo los casi mismos resultados en las diferentes ocasiones, mostrado en la siguiente tabla:

	Memoria Inicial	Memoria en ejecución	Memoria consumida
1	680	617	63
2	685	623	62
3	686	623	63
4	685	623	62
5	685	623	62

Figura 6.4: Tabla de Temperatura

Para poder analizar el estado de la red de los diferentes nodos, se va a mostrar un gráfico utilizando el programa rqt\_graph.

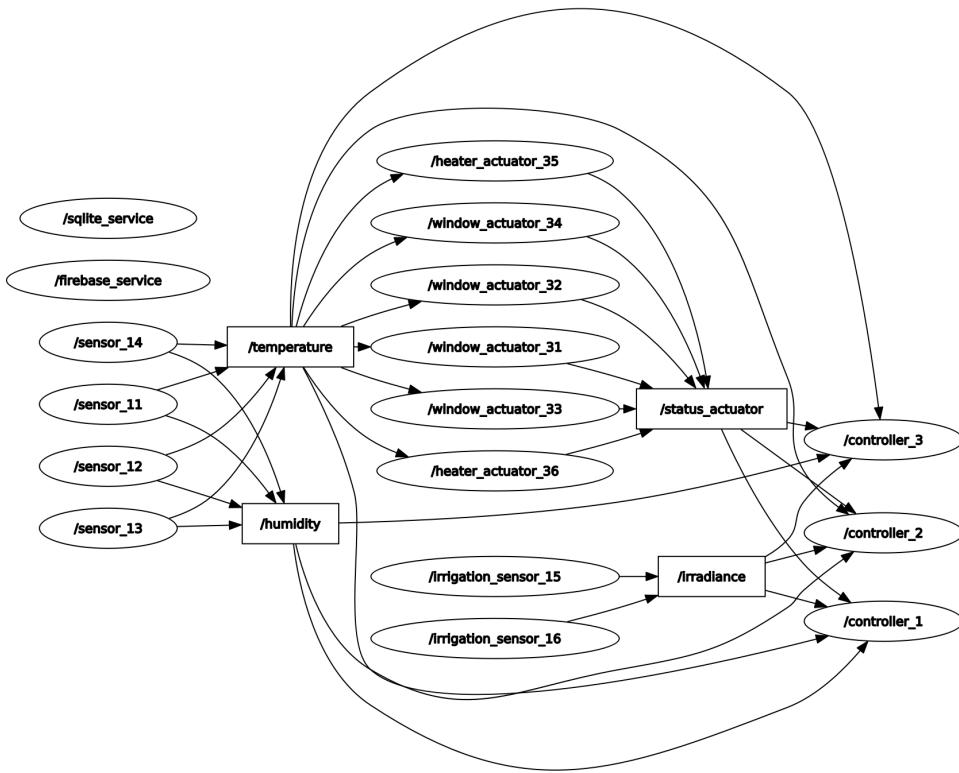


Figura 6.5: Red en el Escenario 3

Por aclarar este escenario, al igual que se ha explicado en el diagrama del análisis 4.6 y en la implementación 5.3, los rectángulos representan los tópicos y los óvalos representan los diferentes nodos.

Para especificar donde se estaba ejecutando cada nodo, se va a listar a continuación:

- Las placas ESP32 están representadas con **sensor\_11** y **sensor\_12**
- La Raspberry Pi 3B utilizada durante todo el escenario está ejecutando los nodos **controller\_3** y **sqlite\_service**
- El portátil Asus esta ejecutando los demás dispositivos de manera virtual.

Finalizando esta sección, se ha utilizado Wireshark [32] para capturar el tráfico generado en la red.

Tras dejar el sistema durante 30 minutos en ejecución, se ha aplicado como filtro que el protocolo utilizado sea RTPS, ya que es utilizado en la comunicación entre los diferentes nodos.

Con la funcionalidad de Stadistics ofrecida por Wireshark, se ha obtenido una gráfica mostrando la cantidad de paquetes cada 10 segundos, representado en la siguiente imagen:

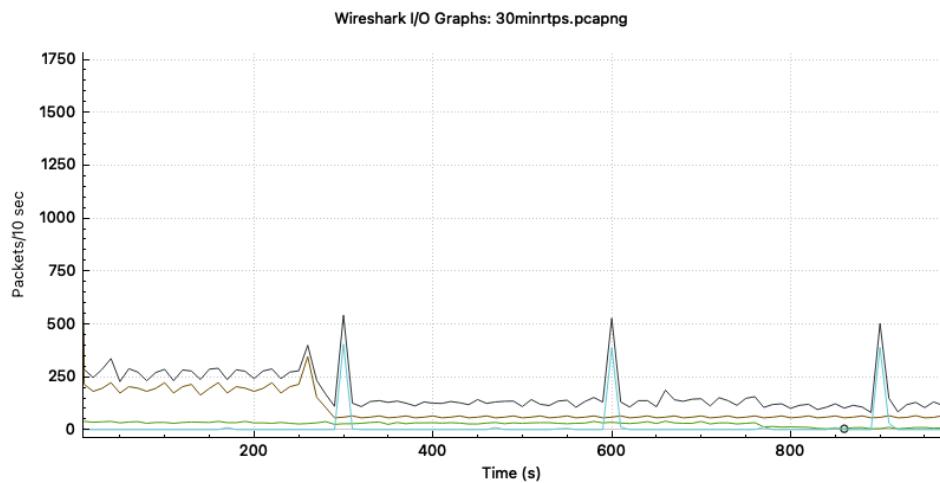


Figura 6.6: Gráfica generada por Wireshark

Como se puede observar, hay varias líneas mostradas en el gráfico:

- La marrón es el protocolo RTPS
- La verde es el protocolo ARP
- La azul es el protocolo TCP
- La negra es el número de paquetes total

Cada cierto tiempo, se produce un pico de paquetes en la red se vuelve a estabilidad. Esto es debido al almacenamiento de los datos en la plataforma de Firebase, que se ejecuta cada ese periodo, establecido con un temporizador.

Por lo tanto se podría concluir que la cantidad de paquetes es estable respecto a RTPS una vez pasado un tiempo desde que comienza la red, y la comunicación con la base de datos cada este periodo produce esos picos.

## Capítulo 7

# Conclusiones y trabajo futuro

En este trabajo se han conseguido cumplir los tres objetivos principales propuestos en la introducción. Entre ellos destacar la posibilidad de añadir diferentes nodos al sistema sin tener que realizar modificaciones o solo las mínima. Esto ha sido posible gracias al haber hecho uso de ROS 2 junto al paradigma Publicador-Suscriptor, únicamente teniendo que definir un publicador o suscriptor nuevo relacionado con los tópicos existentes en esta red.

Tras el estudio de algunas de las herramientas ofrecidas por ROS 2, se han analizado todas las funcionalidades que nos otorga. Además se han aprendido diferentes conceptos relacionados con esta tecnología utilizada en más escenarios de los que se pensaba al principio, y la potencia que tiene al utilizarse en el ámbito de IoT.

Respecto a la interfaz se han podido comprender las bases del funcionamiento del Angular y la estructura de un proyecto de este tipo, diferenciando componentes, módulos y derivados.

Además, gracias a las evaluaciones de los diferentes escenarios, se ha llegado a la conclusión que la Raspberry Pi3B es más que solvente para la ejecución de los nodos requeridos, sin ver influencias claras respecto a los datos publicado en la red producidos por los demás nodos.

Para comentar las tareas que han quedado por realizar se pueden leer en la siguiente sección.

### 7.1. Trabajo futuro

Debido al uso de diferentes tecnologías de las cuales se desconocían su funcionamiento y como trabajar con ellas, tras un periodo de estudio y asimilación de conceptos se ha podido lograr el trabajo anterior. Pero hay puntos por realizar, que en el caso de algunos de ellos, si se hubiese querido implementar todas sus funcionalidades, probablemente daría para otro documento de similar extensión a este.

- **Capa de Seguridad.** Es necesario encriptar la información y la gestión

---

## *CAPÍTULO 7. CONCLUSIONES Y TRABAJO FUTURO*

de los diferentes usuarios en el sistema, añadiendo las tablas que sean necesarias en la base de datos.

- **Extensión de la API REST.** Esta tarea tiene mucho potencial para futura interacción con el sistema, pudiendo acceder a estos datos para ser tratados con distintos fines.
- **Test de carga y despliegue de los dispositivos en un entorno real.** Para ello es necesario un estudio con personal con más conocimientos sobre los diferentes sensores en el mercado y que sería de necesidad en estos escenarios.

# Bibliografía

- [1] ROS 2. About quality of service settings. <https://docs.ros.org/en/foxy/Concepts/About-Quality-of-Service-Settings.html>. Última vez visitada 07 Noviembre 2021.
- [2] ROS 2. Ros 2 documentation: Foxy. <https://docs.ros.org/en/foxy/index.html>. Última vez visitada 06 Noviembre 2021.
- [3] Revista Agropecuaria. Robots agrícolas ayudando a la agricultura. [http://www.revistaagricultura.com/maquinaria-agricola/maquinaria/robots-agricolas-ayudando-a-la-agricultura\\_13780\\_120\\_17170\\_0\\_1\\_in.html](http://www.revistaagricultura.com/maquinaria-agricola/maquinaria/robots-agricolas-ayudando-a-la-agricultura_13780_120_17170_0_1_in.html). Última vez visitada 06 Noviembre 2021.
- [4] AgroSpray. ¿qué es la agricultura 4.0? <https://agrospray.com.ar/blog/agricultura-4-0/>. Última vez visitada 06 Noviembre 2021.
- [5] Angular. Angular. <https://angular.io/>. Última vez visitada 07 Noviembre 2021.
- [6] Clockify. Clockify, time tracking apps. <https://clockify.me/>. Última vez visitada 11 Noviembre 2021.
- [7] Thomas Menzel Daniel Happ, Niels Karowski. Meeting iot platforms requirements with open pub/sub solutions. <https://link.springer.com/content/pdf/10.1007/s12243-016-0537-4.pdf>. Última vez visitada 06 Noviembre 2021.
- [8] Docker. Docker. <https://www.docker.com/>. Última vez visitada 07 Noviembre 2021.
- [9] Eclipse. Cyclone dds. <https://cyclonedds.io/>. Última vez visitada 06 Noviembre 2021.
- [10] eProsimma. Fast dds. <https://www.eprosima.com/index.php/products-all/eprosima-fast-dds>. Última vez visitada 06 Noviembre 2021.

## BIBLIOGRAFÍA

---

- [11] Sigfox España. Agricultura. <https://www.sigfox.es/agricultura>. Última vez visitada 06 Noviembre 2021.
- [12] FastAPI. Fastapi. <https://fastapi.tiangolo.com/>. Última vez visitada 07 Noviembre 2021.
- [13] Fertri. Pantallas térmicas. <https://fertri.com/accesorios-y-componentes/pantallas-termicas/>. Última vez visitada 06 Noviembre 2021.
- [14] Firebase. Documentación firebase. <https://firebase.google.com/docs>. Última vez visitada 07 Noviembre 2021.
- [15] FreeRTOS. Freertos. <https://www.freertos.org/>. Última vez visitada 07 Noviembre 2021.
- [16] Google. Charts. <https://developers.google.com/chart/>. Última vez visitada 07 Noviembre 2021.
- [17] Hispatec. Hortisys. <https://hortisys.es>. Última vez visitada 06 Noviembre 2021.
- [18] Kafka. Kafka. <https://kafka.apache.org/documentation/>. Última vez visitada 06 Noviembre 2021.
- [19] Material. Angular material. <https://material.angular.io/>. Última vez visitada 07 Noviembre 2021.
- [20] micro ROS. Documentación micro-ros. <https://micro.ros.org/>. Última vez visitada 07 Noviembre 2021.
- [21] Nazaríes. Nazaríes, tecnología agrícola. <https://www.tecnologia-agricola.com/nosotros/>. Última vez visitada 06 Noviembre 2021.
- [22] Novagric. Invernaderos capilla. <https://www.novagric.com/es/venta-invernaderos-novedades/tipos-de-invernaderos/invernadero-capilla>. Última vez visitada 06 Noviembre 2021.
- [23] OMG. Data distribution service. <https://www.omg.org/spec/DDSI-RTPS/2.3/PDF>. Última vez visitada 06 Noviembre 2021.
- [24] RabbitMQ. Rabbitmq. <https://www.rabbitmq.com/>. Última vez visitada 06 Noviembre 2021.
- [25] ROS. Ros running on iss. <https://www.ros.org/news/2014/09/ros-running-on-iss.html>. Última vez visitada 06 Noviembre 2021.
- [26] RTI. Connex professional. <https://www.rti.com/products/connex-dds-professional>. Última vez visitada 06 Noviembre 2021.

## BIBLIOGRAFÍA

---

- [27] Alarcontrol S.L. Alarcontrol s.l. <https://www.alarcontrol.com/es/>. Última vez visitada 06 Noviembre 2021.
- [28] SQLite3. Sqlite3. <https://www.sqlite.org/index.html>. Última vez visitada 07 Noviembre 2021.
- [29] Talent. Salario medio para ingeniero informático en españa 2021. <https://es.talent.com/salary?job=ingeniero+inform%C3%A1tico>. Última vez visitada 11 Noviembre 2021.
- [30] Uvicorn. Uvicorn. <https://www.uvicorn.org/>. Última vez visitada 07 Noviembre 2021.
- [31] Vortex. Topics, domains and partitions. <http://download.prismtech.com/docs/Vortex/html/ospl/DDSTutorial/topics-etc.html#:~:text=A%20Topic%20represents%20the%20unit,a%20set%20of%20QoS%20policies>. Última vez visitada 07 Noviembre 2021.
- [32] Wireshark. Wireshark. <https://www.wireshark.org/>. Última vez visitada 07 Noviembre 2021.



# Acrónimos

<b>AMQP</b>	Advanced Message Queuing Protocol
<b>APIs</b>	Application Programming Interfaces
<b>ASGI</b>	Asynchronous Server Gateway Interface
<b>DDS</b>	Data Distribution Service
<b>FreeRTOS</b>	Real-time operating system for microcontrollers
<b>GPRS</b>	Servicio General de Paquetes vía Radio
<b>IA</b>	Inteligencia Artificial
<b>IDL</b>	Interface Description Language
<b>IoT</b>	Internet of the Things
<b>ISS</b>	International Space Station
<b>MIT</b>	Massachusetts Institute of Technology
<b>MoM</b>	Message-oriented Middleware
<b>MQTT</b>	MQ Telemetry Transport
<b>OMG</b>	Object Management Group
<b>QoS</b>	Quality of Service
<b>REST</b>	REpresentational State Transfer
<b>ROS</b>	Robot Operating System
<b>STOMP</b>	Streaming Text Oriented Messaging Protocol
<b>TICs</b>	Tecnologías de Información y Comunicación