UNIVERSITAT POLITÈCNICA DE VALÈNCIA

CAMPUS D'ALCOI

Estudio de viabilidad y despliegue real de una red para IoT

MEMORIA PRESENTADA POR:

Mauro Belda Bou

GRADO DE INFORMÁTICA

Convocatoria de defensa: *Junio de 2020*

Estudio de viabilidad y despliegue real de una red para IoT

Mauro Belda Bou

Resumen
La tecnología actual permite, cada vez más, la creación de redes de dispositivos de sensorización que se utilizan para IoT (del inglés, Internet of Things). Este TFG pretende que se analicen las diversas posibilidades tecnológicas de implementación de estas redes. Además también se ha de presentar un estudio de viabilidad que soporte la tecnología elegida para el despliegue de una red IoT real. El TFG finalizará con la implementación de un ejemplo de uso sobre la red desplegada.
Requisitos técnicos:

- análisis de al menos las alternativas tecnológicas de SigFox y LoRa
- mediciones reales (presupuesto) para el despliegue de la red seleccionada para una prueba de concepto (10 dispositivos IoT)
- mediciones reales (presupuesto) para el despliegue de la red seleccionada para una prueba de concepto (10,000 dispositivos IoT)
- implementación de un ejemplo de uso que valide la red desplegada

Palabras clave: IoT, red, Arduino, LoRa, aplicación.

Summary

Today's technology increasingly enables the networking of sensor devices used for IoT (Internet of Things). This TFG aims at analysing the various technological possibilities for the implementation of these networks. In addition, a feasibility study supporting the technology chosen for the deployment of a real IoT network must be presented. The TFG will end with the implementation of an example of use on the deployed network.

Technical requirements:

- analysis of at least the technological alternatives of SigFox and LoRa
- actual measurements (budget) for the deployment of the network selected for a proof of concept (10 IoT devices)
- actual measurements (budget) for the deployment of the network selected for a proof of concept (10.000 IoT devices)
- implementation of a usage example that validates the deployed network

Keywords: IoT, network, Arduino, LoRa, application.

Dirección

Pau Micó, Carlos Sastre

Contenidos

1 Introducción
- 1.1 Antecedentes
- 1.2 Objetivos
- 1.3 Requerimientos

2 Anteproyecto
- 2.1 Estado del arte
  - 2.1.1 Backend de la red IoT
  - 2.1.2 Entornos de desarrollo y lenguajes de programación
  - 2.1.3 Servidor de bases de datos
  - 2.1.4 Broker MQTT
  - 2.1.5 Casos similares
    - 2.1.5.1 Amazon
    - 2.1.5.2 Field Gateway Client
    - 2.1.5.3 LoRaWAN using Python, Zenyth and TTN
- 2.2 Estudio de propuestas
  - 2.2.1 Propuesta 1
  - 2.2.2 Propuesta 2
- 2.3 Justificación
  - 2.3.1 Estimación de recursos
  - 2.3.2 Impacto económico
  - 2.3.3 Propuesta final

3 Implementación
- 3.1. Implementación de las partes de la red
  - 3.1.1 Configuración del gateway
  - 3.1.2 Configuración del backend
  - 3.1.3 Configuración de la controladora
  - 3.1.4 Configuración del broker MQTT
    - 3.1.4.1 Conexión backend-broker
    - 3.1.4.2 Conexión broker-Python
  - 3.1.5 Configuración de la conexión Python-Base de datos
  - 3.1.6 Diseño de la base de datos
  - 3.1.7 Alternativa a controladora MKR WAN 1300
    - 3.1.7.1 Dispositivos
    - 3.1.7.2 Scripts de funcionamiento
    - 3.1.7.3 Imágenes de los componentes
    - 3.1.7.4 Registro en el backend
- 3.2 Entorno de desarrollo de la aplicación

3.3 Implementación práctica de la aplicación
3.3.1 Maquetas
3.3.2 Diagramas de funcionamiento
3.4. Pruebas
3.4.1 De sistema
3.4.2 De integración de sistemas
3.4.3 De volumen

4 Resultados
4.1 Migración al entorno de producción
4.2 Manual del usuario
4.2.1 Instalación de la aplicación
4.2.2 Explotación

5 Conclusiones
5.1 Conclusiones personales
5.2 Futuras líneas de desarrollo

6 Bibliografía

7 Acrónimos

8 Anexos
8.1 Codigos

1 Introducción

1.1 Antecedentes

El proyecto estaría dentro del área de redes IoT, y el principal motivo por el que se ha decidido realizarlo, es por el creciente auge y demanda del sector Internet of Things, ya que el uso de sensores en la tecnología es creciente. De esta manera, se ofrece la posibilidad de reunir en una sola red las mediciones de distintos sensores distribuidos por distintos espacios físicos.

1.2 Objetivos

Voy a centrar este desarrollo en:

- Almacenar los datos de las mediciones de los distintos sensores en una plataforma
- Visualización y gestión de todas las mediciones de manera centralizada
- Comparar distintas tecnologías y componentes para el despliegue de una red IoT
- Despliegue de una red para la centralización de datos de sensores distribuidos geográficamente.

1.3 Requerimientos

- Backend para albergar una red de dispositivos IoT.
- Sensores de distintos tipos, controladoras donde realizar las conexiones, material físico de red y material adicional de conexionado, entre otros.
- Broker de mensajería para el intercambio de mensajes entre componentes de la red.
- Posibles scripts de programación para recogida y tratado de datos.
- Base de datos para el almacenado de las mediciones.
- Aplicación modular. Los diferentes módulos de gestión se podrán desarrollar independientemente y su integración en la aplicación no dependerá de su estado.
- Se desarrollaran los siguientes módulos de la aplicación:
    - Control del acceso a los datos mediante contraseña almacenada en base de datos. Se verifica la confidencialidad de datos
    - Consulta de información de las mediciones de los sensores
    - Gestión de los contenidos de la base de datos, con posibilidad de eliminar documentos (registros) de la base de datos
    - Categorización de los sensores por tipo de sensor.

2 Anteproyecto

2.1 Estado del arte

2.1.1 Backend de la red IoT

- **Loriot:** es una empresa que ofrece la posibilidad de desplegar soluciones IoT de alta distancia. Ofrecen una estructura para el desarrollo de redes LoRaWAN y un software de fácil uso para la gestión de la red. Entre los distintos servicios que ofrecen podemos situar [1] [2]:
    - **Network Server:** es la parte central de una red LoRa, que recibe los paquetes enviados por el gateway de nuestra red. Actúa como intermediario entre los nodos finales y el servidor de aplicaciones.

      Se encargará de realizar la deduplicación de los paquetes que le llegan (elimina paquetes redundantes), así como de realizar comprobaciones de seguridad antes de enviar la información al servidor de aplicación. Además, será el encargado de identificar la información recibida de los nodos finales para saber que solo almacenaremos en el servidor de aplicación información de nodos de nuestra red, y no de otros nodos que podrían estar transmitiendo datos a nuestras gateways.

      Puede gestionar las velocidades de transferencia y la potencia de salida (radiofrecuencia) de los nodos finales conectados a nuestra red, reduciendo así la congestión de nuestra red en momentos puntuales.

    - **Application Server:** es el principal encargado de desencriptar la información que le envía el servidor de red y enviarla a las salidas configuradas en el backend.

      Permite crear una aplicación y registrar, gestionar y organizar los dispositivos finales como dispositivos a esta misma. Así mismo, permite visualizar los últimos paquetes de datos recibidos, a modo informativo. También permite visualizar otros datos como la señal de recepción de los mensajes, gráficos informativos, etc.

      Con la cantidad de datos a procesar, se requiere almacenar la información en bases de datos, por lo que permite configurar salidas de distintos tipos. Entre ellas, destacan bases de datos como Microsoft Azure o brokers como MQTT, entre otros.

- **Join Server:** Loriot ofrece una opción mediante la cual automatizar el proceso de registro y conexión de nuestros dispositivos IoT a nuestra red LoRa. Este servicio se usa en entornos de producción, donde no podemos asignar manualmente todos los dispositivos a la red. Esta herramienta sería necesaria en el desarrollo de la red de producción de este proyecto.

Existen ciertos identificadores importantes en este backend [2]:

- **Identificadores de una red Loriot:** para el correcto funcionamiento de una red Loriot, se usan diversos identificadores o keys que dotan a la red de un mayor nivel de seguridad. Podemos destacar los siguientes:
  - **End-device address (DevAddr):** identifica un nodo final dentro de la red LoRa. Tiene un tamaño de 32 bits.
  - **Application identifier (AppEUI):** identificador global de la aplicación que es almacenado en el nodo final antes de que este forme parte de la red.
  - **Network session key (NwkSKey):** identificador específico para el nodo final. Es usado tanto por el nodo final como por el servidor de red para verificar la integridad de los mensajes enviados en los paquetes. También sirve para encriptar y desencriptar la carga útil de los paquetes usados a nivel de MAC.
  - **Application session key (AppSKey):** identificador específico para el nodo final. Es usado tanto por el nodo final como por el servidor de aplicación para encriptar y desencriptar la carga útil de los paquetes de datos. Esta carga útil es encriptada de modo que solamente el nodo final y el servidor de aplicación puedan consultarla. No obstante, no verifica la integridad de los datos, por lo que un servidor de red por el que pasase el paquete podría alterar la carga útil del mismo. Para solucionar este problema, se establecen lazos de confianza entre servidores de red y de aplicación.

Se muestran distintas licencias según el propósito y el tamaño del despliegue de la red. Este backend ofrece las siguientes [3]:

- Licencia **Community Public Network Server:** soporta de manera gratuita 1 gateway y 10 dispositivos, por lo que es ideal para desplegar una red LoRa inicial, no enfocada a un entorno de producción. Permite recibir mensajes ilimitados diarios así como soporte técnico básico.
  Proporciona un servidor público y compartido, es decir, los recursos del servidor de red son compartidos por diversos usuarios.

- Licencia **Private Network Server:** proporciona un registro ilimitado de gateways y dispositivos en una red, así como despliegue de nube a nivel mundial. Es ideal para entornos IoT enfocados a una alta producción.
  Proporciona un servidor privado, es decir, con todos sus recursos dedicados a nuestro proyecto, sin compartición de estos mismos con otros clientes.
- Licencia **Professional Public Network Server:** proporciona un registro ilimitado de gateways en una red, así como despliegue de nube a nivel mundial. Soporta el registro de hasta 2500 dispositivos, con servidores con poca latencia y número ilimitado de mensajes.
  Es ideal para entornos IoT con un número de dispositivos elevado, enfocado a producción.

- **The Things Network:** a diferencia de Loriot, TTN es un backend totalmente gratuito, sin licencias de pago. Su uso es muy extendido entre usuarios particulares que requieren de un backend sencillo para su entorno IoT. Normalmente, no se usa para entornos de producción. Soporta conexiones de entre 5-15 km con poco ancho de banda (51 bytes/mensaje). A continuación, se detallarán sus partes principales:
  - **Red [4]:**
    - **Network Server:** parte responsable de la funcionalidad específica de LoRaWAN.
    - **Router:** dispositivo responsable de gestionar el estado de los gateway y de programar las transmisiones de datos. Cada uno de estos se conecta a uno o más brokers.
    - **Broker:** parte principal de la red del backend. Su deber es el de mapear un dispositivo a su aplicación. También encaminan mensajes de descarga al router al que van dirigidos, así como encaminar mensajes de subida a la aplicación destino.
    - **Handler:** es la parte responsable de gestionar los datos de las aplicaciones. Para ello, se conecta a un broker donde registrar aplicaciones y dispositivos. Además, es el punto en el cual los datos son encriptados y desencriptados.
    - **Discovery:** tipo de servidor que almacena las referencias de los brokers, handlers y los routers. Si alguna parte del backend necesita localizar alguna parte de las anteriormente mencionadas, solo debe acudir a este apartado. Centraliza la descentralizada estructura que conforma la red del backend.
  - **Aplicaciones [5]:** parte del backend donde se referencian los dispositivos registrados en el mismo.
  - **Gateways [6]:** forman un puente de conexión entre los dispositivos y el backend. Usa redes WiFi o Ethernet para realizar la conexión al backend. Todos los gateway dentro del rango de un dispositivo recibirán los mensajes que envía el

dispositivo, y los encaminarán al backend. Un solo gateway puede encaminar datos de cientos de dispositivos.

- **Dispositivos [7]:** son las controladoras o dispositivos finales que son registrados en el backend. Al ser registrados, pueden intercambiar datos con el mismo.

A pesar de ser gratuito, este backend ofrece diversas medidas de seguridad para securizar de esta manera los datos de los usuarios. Entre ellas, destaca la encriptación AES de 128-bits de la carga útil de los paquetes de datos. Esta encriptación se realiza entre el nodo que envía los datos y el handler del backend. Tanto el router como el broker no pueden desencriptar datos.

En este caso, se ha optado por usar el backend que ofrece Loriot, ya que ofrece mejores posibilidades para desplegar un entorno de producción IoT. Sus licencias gratuitas pueden desplegar el entorno de 10 dispositivos práctico, y sus licencias de pago pueden hacer frente al entorno de 10,000 dispositivos teórico de este proyecto.

2.1.2 Entornos de desarrollo y lenguajes de programación

En la actualidad, se pueden usar diversos lenguajes y entornos de programación para el desarrollo de aplicaciones multiplataforma o páginas web.

Se van a abordar diversas opciones de desarrollo en la actualidad, exponiendo sus bases y ciertas ventajas y desventajas de cada una de ellas.

- **Flutter [8]:** es un kit de herramientas de interfaces de usuario creado por Google. Se usa para realizar aplicaciones con buen diseño y compiladas nativamente. Estas aplicaciones son multiplataforma, es decir, válidas y escalables para móvil, web y escritorio, desde un solo fichero de código.

  El framework Flutter usa el lenguaje de programación Dart, un lenguaje desarrollado por Google orientado al desarrollo web y de aplicaciones. Nace como alternativa a lenguajes típicos como HTML, CSS y Javascript.

  Esta plataforma se caracteriza por los siguientes puntos:
  - Desarrollo rápido: ofrece “recarga en caliente” del código. Esto da la posibilidad de realizar cambios en la aplicación y poder observarlos en el emulador casi al instante, sin necesidad de recompilar la aplicación. Este apartado supone un ahorro de tiempo en el desarrollo de una aplicación, y que no todos los frameworks ofrecen.
  - Interfaces de usuario atractivas: posee un gran catálogo de *widgets* o elementos prediseñados que pueden ser usados de una manera sencilla en la aplicación de

un usuario. Muchos de ellos son totalmente personalizables, lo que ayuda a reducir costos en un diseño elaborado.
    - Rendimiento nativo: todos los *widgets* o elementos son totalmente compatibles con las distintas plataformas móviles, tales como iOS y Android. Esto facilita el desarrollo de una sola aplicación con un único diseño válido para todas las plataformas móviles.

- **Android Studio [9]:** entorno de desarrollo integrado oficial para el desarrollo de aplicaciones para el sistema Android.

  Android Studio permite el uso de diversos lenguajes de programación para nuestras aplicaciones, aunque hay dos que son los más usados por los desarrolladores: Java y XML.

  XML es un lenguaje sencillo usado para formar el diseño de la interfaz de la aplicación, que puede ser autogenerado si el usuario arrastra componentes ya diseñados a su aplicación, o bien escrito por el propio usuario. Con el editor de Android Studio, el usuario puede previsualizar los resultados en tiempo real.

  Java es el lenguaje usado para dar funcionalidad a la aplicación y, de esta manera, conseguir el funcionamiento global de todos los componentes de la misma.

  No obstante, existen otras alternativas para el desarrollo de la misma, como C++ o Kotlin entre otros.

  Ofrece las siguientes características:
    - Sistema de compilación flexible basado en Gradle, que es una herramienta que permite la automatización de la compilación de código de una manera eficiente y flexible.
    - Emulador rápido y con multitud de funciones.
    - Entorno unificado donde poder desarrollar aplicaciones para todos los dispositivos Android.
    - Ejecución de cambios en la aplicación sin necesidad de reiniciar la misma.
    - Compatibilidad de nuestra aplicación con Google Cloud Platform o GitHub.
    - Editor de diseño que permite a los usuarios arrastrar y soltar componentes de la interfaz, sin necesidad de código.

Para este proyecto, se va a usar el framework Flutter para el desarrollo de la aplicación donde visualizar los datos. Este framework es flexible y sencillo de usar, además de ser una herramienta novedosa con multitud de funciones. Cabe destacar su biblioteca de elementos ya diseñados es una opción que facilita el desarrollo rápido de una aplicación, junto con la recarga en caliente.

2.1.3 Servidor de bases de datos

En cuanto al almacenamiento de datos, existen diversos tipos de bases de datos muy empleadas en la actualidad. Entre ellas destacamos [10]:

- **Bases de datos relacionales:** son aquellas bases de datos que se rigen por el modelo ACID. Podemos destacar una serie de ventajas y desventajas de las mismas:
  - *Ventajas:*
    - Documentación extensa y estándares bien definidos y aceptados
    - Debe cumplir el modelo ACID, el cual dota de seguridad y consistencia a la propia base de datos.
  - *Desventajas:*
    - Poco escalables, ya que siguen de manera estricta el modelo de datos de cada tabla
    - Si el modelo de datos no es lo suficientemente robusto, una migración de sistema gestor puede ser costosa.
- **Bases de datos no relacionales:** alternativa a las relacionales. Funcionan sin un esquema exacto de almacenado, y sin relaciones entre los distintos documentos que las conforman. Destacamos las siguientes ventajas y desventajas:
  - *Ventajas:*
    - Escalables, ya que no siguen un esquema de datos exacto
    - Son tolerantes a los fallos, ya que no siguen un modelo
    - Las migraciones entre sistemas son relativamente sencillas.
  - *Desventajas:*
    - Al ser relativamente recientes, la documentación y los estándares son más escasos que las relacionales
    - Al no seguir esquemas fijos, cada base de datos sigue sus métodos propios para formatear sus datos.

Una vez expuestos los dos tipos principales de bases de datos, se hará referencia a las bases de datos más conocidas de cada tipo, y se escogerá una de ellas para el almacenamiento del proyecto, según convenga.

- **Cloud Firestore de Firebase - No relacional [11]:** base de datos NoSQL flexible, escalable y en la nube a fin de almacenar y sincronizar datos. Se puede emplear para la programación en servidores, dispositivos móviles y web.
  Almacena los datos en documentos que contienen campos, a los cuales se asignan los valores. Estos documentos se almacenan en colecciones, que no son más que contenedores que organizan los datos. Estos documentos admiten distintos tipos de campos, desde strings y números hasta objetos anidados complejos.
  Destaca por los siguientes aspectos:
  - Flexibilidad: admite estructuras de datos flexibles y jerárquicas gracias a su modelo de datos.

- Consultas expresivas: consultas flexibles y con filtros. Diversos filtros se pueden incluir en una misma cadena de manera sencilla.
- Actualizaciones en tiempo real: sincronización de datos para la actualización de datos de cualquier dispositivo conectado.
- Ajustable a la escala de datos: replicación automática de datos multiregión, operaciones atomáticas por lotes, asistencia real sobre transacciones, etc.

- **MySQL - Relacional [12]:** base de datos SQL que almacena datos en distintas tablas siguiendo un esquema predefinido. Cada tabla consta de registros o filas, y cada registro de atributos o columnas. En cada tabla, todos los registros tienen los mismos atributos. Existen relaciones entre las distintas tablas, así como claves primarias y ajenas que aseguran la consistencia de datos de las mismas.
  Este tipo de base de datos debe ser alojada en un servidor propio o en un hosting externo para poder funcionar. No obstante, se puede hacer uso de MySQL Server en una máquina propia. Destaca por los siguientes aspectos:
  - Open Source: el software MySQL es gratuito y puede ser modificado según necesidades.
  - Servidor MySQL: es rápido, escalable, seguro y fácil de usar. Esto se debe a que requiere de pocos recursos para funcionar, y a que puede funcionar sobre un gran número de máquinas.
  - Ampliamente soportado: el software de MySQL soporta múltiples backends, así como programas cliente, librerías o APIs. Se puede enlazar de manera sencilla a una aplicación.

Para este proyecto, se ha decidido usar los servicios de almacenamiento de Cloud Firestore de Firebase debido a su sencillez de uso y despliegue. Además, ofrece tarifas base gratuitas para proyectos IoT pequeños y medianos. Incluye también el hosting, por lo que no nos debemos de preocupar por buscar uno alternativo.

2.1.4 Broker MQTT

Como salida de datos del backend, se pueden configurar diversos servicios, que pueden ser de terceros o estándar. En el caso de configurar una salida estándar, una opción muy usada es configurar un broker con el protocolo MQTT.

Hay diversas empresas que ofrecen este servicio de manera gratuita, y fácil de configurar para el usuario. Este es el caso de empresas como CloudMQTT.

- **CloudMQTT [13]:** es una empresa dedicada a la gestión de servidores Mosquitto en la nube.
  Hacen uso del protocolo MQTT, un protocolo de mensajería máquina a máquina que, en este caso, hace uso del patrón de comunicación publish-subscribe. Este protocolo

realiza un buen desempeño en sistemas IoT que se caracterizan por ser poco pesados, donde se transmiten paquetes de datos de pequeño tamaño, de apenas unos bytes.
CloudMQTT presenta distintos tipos de licencias, que van desde la gratuita, hasta diversas de pago que ofrecen un número de recursos limitado. Presenta, además, diversos servicios que facilitan al usuario las consultas de los mensajes que se publican en tópicos, así como opciones de seguridad, como ACLs (listas de control de acceso). Destacamos los siguientes servicios:

- Certificados: permiten establecer dominios propios a nuestras instancias del broker, para referirnos a ellas más tarde.
- Usuarios y ACLs: permite establecer listas de control de acceso a usuarios que deseemos crear. Cada usuario tendrá unos permisos R/W específicos en cada instancia del broker.
- Puentes: cliente MQTT embebido en el broker que permite conectar y enviar mensajes a otros brokers.
- AWS Kinesis: permite conectar la salida del broker MQTT a los servicios de Amazon Kinesis para la recepción de los datos.
- Websocket: emula en el navegador un cliente MQTT. De esta manera, se pueden publicar mensajes en un tópico o ver los mensajes que se reciben en el propio broker en tiempo real. Esta es una opción para licencias gratuitas que es útil para realizar pruebas de funcionamiento del sistema.
- Otras: como información de las conexiones actuales al broker, Logs o estadísticas.

Un punto a destacar de CloudMQTT es que solo nos ofrece el servicio de un broker, que no es más que recibir datos, tratarlos, y posteriormente enviarlos a otro punto que se encargará de su recogida. Por tanto, se debe de crear una salida para los datos del broker, que suele ser un script. Este script puede estar disponible en lenguajes como Python, Java, NodeJS, etc.

En el caso de querer utilizar un broker IoT que de facilidades y opciones a la hora de crear salidas a bases de datos, deberemos usar brokers de terceros, como Amazon AWS, entre otros.

- **Amazon AWS IoT [14]:** se presenta como una plataforma en la nube que ofrece cientos de servicios integrales de centros de datos. Entre ellos, destacan servicios relacionados con las redes IoT, que permiten conectar dispositivos IoT entre ellos, securizar los datos y las interacciones o procesar datos entre otros.
  Permite, además, redirigir los datos de dispositivos a una de las salidas de Amazon AWS configurada, para que posteriormente puedan ser, entre otras, almacenados. Destacan por los siguientes puntos:
  - Conexión y administración de dispositivos: admite conectar cualquier número de dispositivos a la nube, así como el uso del protocolo MQTT para una comunicación ligera.

- Protección de datos: conexiones de dispositivos a Amazon AWS cifradas extremo a extremo. También ofrece políticas de acceso a aplicaciones con permisos específicos.
- Procesamiento y utilización de datos de dispositivos: permite filtrar, transformar y utilizar los datos de los dispositivos según necesidades del cliente. Permite el uso de otros servicios de Amazon para una mayor comodidad.
- Lectura de dispositivos en cualquier momento: nos ofrece la posibilidad de usar datos de ciertos dispositivos aunque el estado de los mismos sea desconectado.

Amazon AWS IoT ofrece la creación de salidas de datos desde la instancia del broker creado. Da la posibilidad de crear bases de datos MySQL, o MariaDB entre otras, en su mayoría relacionales. De esta manera, el cliente puede dejar de preocuparse por el procesamiento de datos y la redirección de los datos generados por el broker.
En el caso de MySQL, el cliente solo tiene que configurar un cliente MySQL (que puede ser desde la herramienta MySQL Workbench), y diversas opciones de enlace desde Amazon AWS.

Después de exponer dos tipos de brokers de intercambio de mensajes, se ha llegado a la conclusión de usar el broker ofrecido por CloudMQTT. Esto se debe a que ofrece una mayor flexibilidad a la hora de tratar los datos salientes del mismo broker, puesto que somos nosotros los que configuramos un script para la toma y procesado de los datos. Las diversas opciones de procesado de datos a partir de entornos y lenguajes de programación se define en el punto 2.1.2 Entornos de desarrollo y lenguajes de programación.

2.1.5 Casos similares

En este apartado se van a introducir diversos ejemplos IoT similares al actual proyecto que se está desarrollando. Existen diversas empresas u organizaciones que actualmente impulsan el desarrollo de la tecnología LoRa para facilitar tareas y tratar de unificar dispositivos IoT en una red centralizada.

Entre los ejemplos se habla del proyecto Sidewalk de Amazon, el despliegue de un cliente LoRa básico, así como el desarrollo de una red LoRa con conexión a backend. Este último ejemplo es muy similar al proyecto que se está desarrollando en este documento.

2.1.5.1 Amazon

Existen empresas que en los últimos años han desarrollado proyectos relacionados con dispositivos IoT. Una de estas empresas es Amazon, que ha desarrollado un proyecto llamado Sidewalk [15].

Sidewalk es un proyecto que parte de una idea inicial de poder desarrollar una red inalámbrica de largo alcance (similar a una red WI-FI, pero con un alcance mucho mayor) para poder

interconectar dispositivos IoT, incluso estando estos mismos a mucha distancia del nodo de conexión más cercano.

Este proyecto ha sido enfocado inicialmente para uso doméstico, es decir, poder controlar ciertas funcionalidades de los hogares a través de dispositivos IoT (como sensores para facilitar y automatizar ciertas tareas). Podemos destacar el uso de sensores para automatizar el riego de un jardín, controlar el encendido de luces, etc.

No obstante, es algo que se quiere llevar un paso más allá, es decir, poder extender este tipo de red a los exteriores de los hogares, de tal manera que podamos disfrutar de los servicios que ofrece la misma en cualquier parte de nuestra ciudad. Como ejemplo práctico podríamos localizar a nuestra mascota si ésta se perdiese (en el caso de que la red IoT cubriera una zona muy extensa).

En cuanto a especificaciones técnicas, Sidewalk trabaja a 900 MHz (frecuencia para ondas radio permitida en Estados Unidos, frente a 868 MHz en Europa), lo que hará que la señal se atenúa poco a mayores distancias recorridas o a elementos que deba atravesar. Al trabajar con tan bajas frecuencias, el consumo energético de estos dispositivos es muy bajo, por lo que la batería de los mismos puede durar incluso meses.
Otra característica de trabajar a frecuencias bajas es el poco ancho de banda de esta red, aunque precisamente no es algo necesario para los dispositivos IoT. Estos dispositivos envían señales como, por ejemplo, encender o apagar luces, activar o desactivar el riego, etc.
Por tanto, podemos decir que lo que realmente nos interesará de este tipo de redes es la gran cobertura que ofrecen.

Un punto interesante de este proyecto es la denominada Sidewalk Mesh, que se basa en interconectar de forma automática los puntos de acceso de los distintos hogares de las ciudades para formar así una gran red IoT que proporcione kilómetros de cobertura.
La propia empresa ya ha repartido 700 dispositivos a determinados hogares para comprobar el rendimiento de esta idea, y aseguran que si todo va según lo esperado, lanzarán este proyecto oficial en el año 2020 a modo de protocolo de red.

No obstante, Amazon no ha sido la única empresa en lanzar un proyecto de este tipo. Hay empresas que desarrollaron protocolos de red inalámbricos de baja potencia y alta cobertura mucho antes que lo hiciera Amazon. Unos ejemplos claros serían Zigbee, LoRa o Dect, cada uno con sus características propias.

2.1.5.2 Field Gateway Client

En este proyecto se desarrolla un cliente básico LoRa que capta la información del sensor de temperatura, para su posterior procesado y envío a un servidor [16].

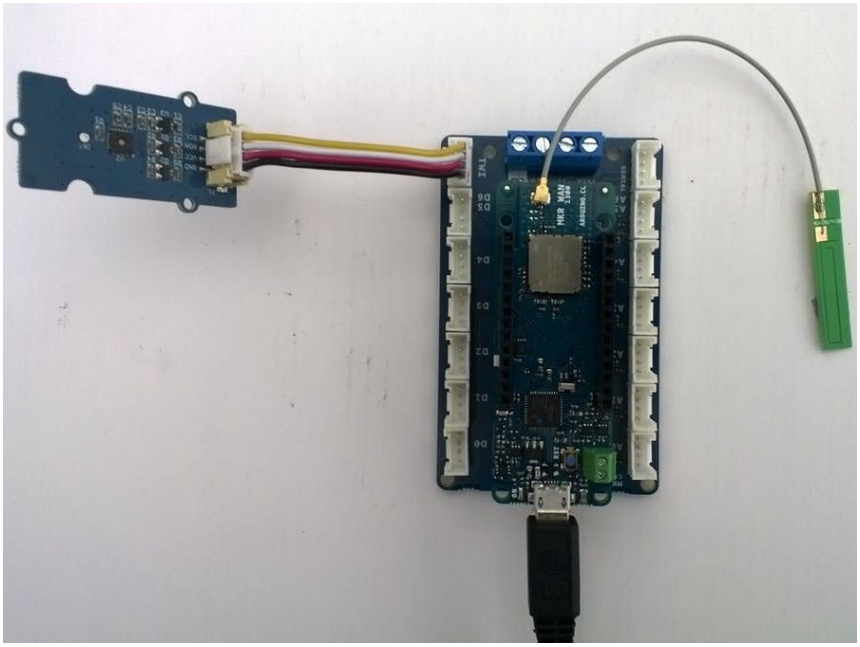


Figura 1. *Cliente básico LoRa.*

Para el desarrollo del cliente, solamente se requiere:

- Arduino MKR WAN 1300
- Arduino MKR Connector Carrier (estilo GROVE)
- Sensor de temperatura
- Antena LoRa

El funcionamiento es simple. Se conecta el Arduino GROVE por puerto USB a nuestro ordenador, y el sensor mediante un cable Grove de 4 pines a la misma placa. De esta manera, la controladora GROVE recibirá los datos analógicos de temperatura que le capte el propio sensor.

A continuación, se conecta la placa con conectividad LoRa (MKR WAN 1300) a la controladora GROVE, para que esta misma reciba los datos recibidos por la controladora GROVE. La controladora LoRa, finalmente, será la que procese y envíe a través de la red (mediante wireless, con la antena conectada a la misma) los datos de temperatura a nuestro backend.

La versatilidad del protocolo LoRa permite realizar distintos montajes con los mismos o similares propósitos, con una variedad amplia en la elección de dispositivos de conexión.

2.1.5.3 LoRaWAN using Python, Zerynth and TTN

Este es un ejemplo de un despliegue completo de una red LoRa, compuesta por los siguientes dispositivos [17]:

- Nodos finales (MCU board named Flip&Click and Microchip RN2483)

- Gateway (Link Labs BS-8)
- Backend (The Things Network)
- IDE de desarrollo de los nodos finales (Zerynth Studio)

En primer lugar, se debe configurar el gateway. Luego, se conecta mediante cable ethernet el gateway a un ordenador personal, y se accede vía navegador a la dirección IP del mismo.
Acto seguido, se configura la conexión LoRa, seleccionando como packet forwarder a TTN.

Una vez configurado el gateway, se añade a la red TTN el gateway ya configurado. Para ello, desde TTN, se debe seleccionar la opción **Get started by adding one,** donde se debe introducir, entre otros datos menos relevantes, el identificador EUI del gateway.

El siguiente paso es construir una aplicación desde TTN para la recepción de datos de los nodos finales. Hay que registrar los nodos finales en esta aplicación. Como datos importantes, se deben introducir el Device ID y el Device EUI de los nodos finales (que se encuentran en las controladoras).

Finalmente, habrá que configurar los nodos finales desde el IDE Zerynth, con el lenguaje de programación Python. Se puede configurar códigos simples para conseguir los identificadores de los nodos anteriormente mencionados.
No obstante, el módulo LoRa Click deberá estar conectado sobre la placa Flip Click (similar a una placa Arduino UNO o DUE). El módulo LoRa Click integrará la conectividad LoRa en el sistema.

Una vez realizado el paso anterior, se ejecuta una porción de código para el envío de datos simples, de tal manera que se prueba la correcta comunicación entre nodos finales y el backend. Si todo ha funcionado correctamente, se podrán ver los datos enviados desde las controladoras.

El esquema básico es el de la figura 2, donde se incluye todo el material necesario definido al inicio del ejemplo.

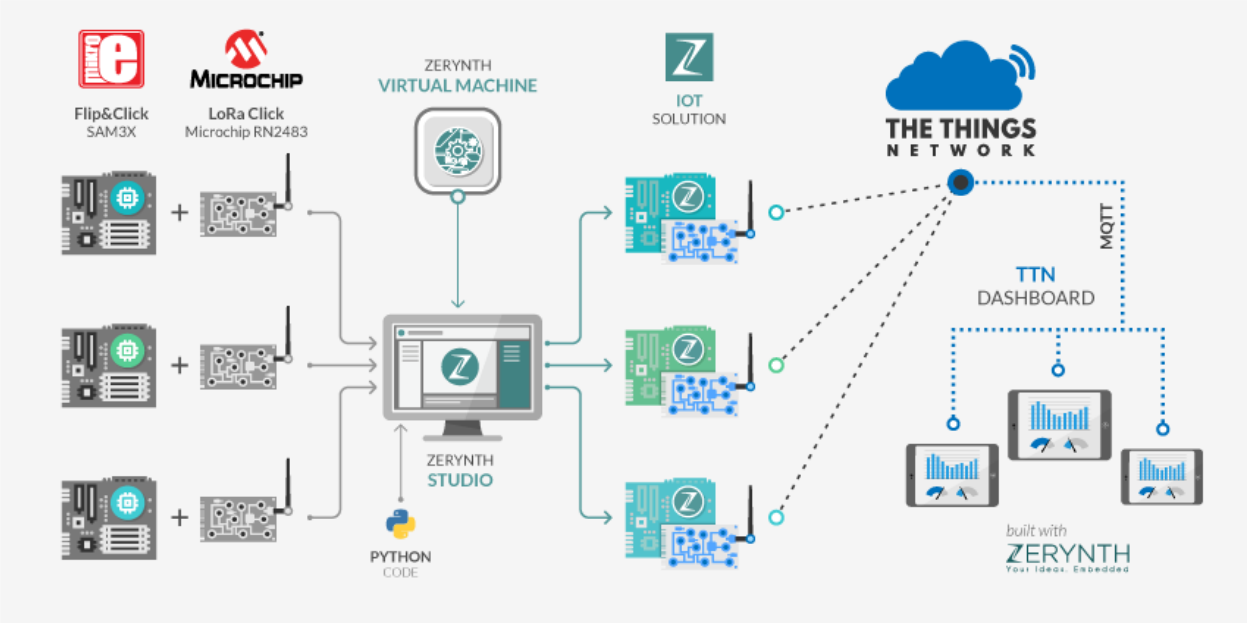


Figura 2. *Esquema de red IoT de ejemplo.*

Este ejemplo es un proyecto muy similar al que se está realizando en este TFG, solo que en el proyecto, se usarán dispositivos un poco distintos, así como el backend Loriot. El esquema de funcionamiento será muy similar.

2.2 Estudio de propuestas

2.2.1 Propuesta 1

En esta propuesta se propone el uso de la tecnología SigFox para el desarrollo de una red IoT [18] [19].

SigFox se caracteriza por proporcionar al cliente su propia arquitectura, es decir, ofrece al cliente un servicio de backend de la propia empresa (SigFox cloud y gateways, entre otras partes). Este backend es el encargado de procesar y almacenar la información de los nodos finales de la red, con el fin de desplegarla, si se quiere, sobre una aplicación.

La tecnología trabaja con ondas radio, es decir, ondas a frecuencias de 868 MHz (en Europa, varía según países), por lo que permite desplegar redes de alto alcance geográfico (varios kilómetros en situaciones ideales, como en campo abierto) a baja potencia (los dispositivos IoT que desplegaremos no requieren de una gran tasa de transferencia de datos).
Por esto, el consumo de una red de este tipo es realmente bajo, ya que una pila/batería puede durar diversos años sin problema gracias a protocolos que permiten el envío de mensajes muy pequeños.

Impone ciertas restricciones de uso, entre las cuales podemos destacar:

- Un máximo de envío de 140 mensajes (de 8 a 12 bytes) por dispositivo por día
- Servicios de localización en dispositivos (al activarlos se encarece el precio de la licencia en unos 2€ por dispositivo).

Como ventajas principales, ofrece acceso a su servicio de nube (backend), soporte online al cliente y cobertura internacional.

Pese a trabajar a bajas velocidades de transferencia por trabajar con ondas radio, puede ofrecer transferencias de 100 a 600 bits/s, dependiendo de la región y circunstancias. Opera en la banda publica de 200 kHz (Ultra Narrow Band radio modulation), ocupando cada mensaje 100Hz de la misma. Ofrece, además, una gran resistencia a interferencias al realizar comunicaciones a grandes distancias.

Cabe destacar que ofrece una interfaz web desde la cual poder dar de alta dispositivos en la red, así como realizar distintas configuraciones sobre la misma. También da la posibilidad de conectar una aplicación propia para la recogida de datos de los nodos finales.

Podemos decir que SigFox se caracteriza por sus dispositivos de bajo consumo, larga duración de las baterías de los mismos, bajo coste por dispositivo, alta capacidad de red y rango de conexión elevado.

Esquema de red Sigfox [19]

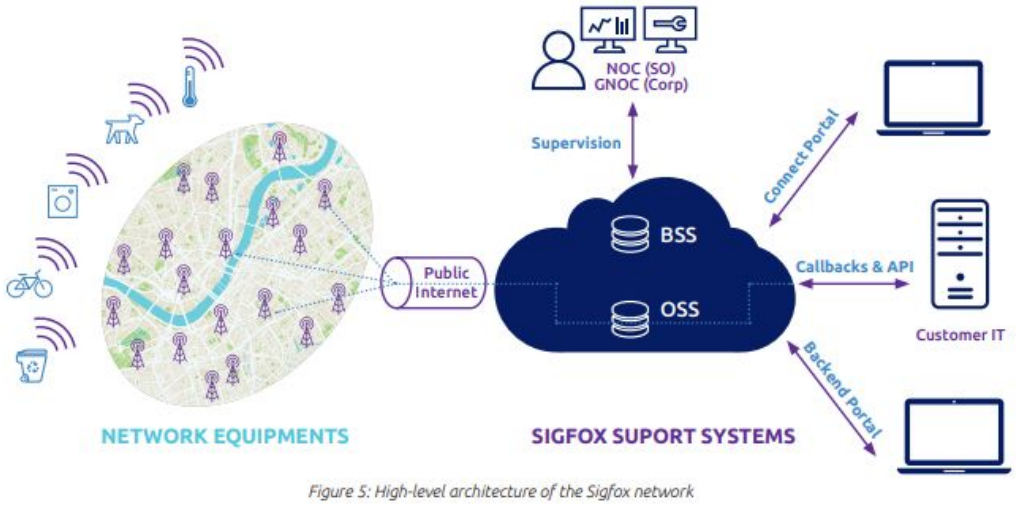


Figure 5: High-level architecture of the Sigfox network

Figura 3. *Esquema de red Sigfox.*

En el apartado "Network Equipments" de la figura 3, se pueden apreciar los distintos gateway desplegados por SigFox en distintas ubicaciones. A estos gateway se dirigirán los distintos sensores emitiendo datos a ciertas frecuencias.

Una vez un gateway recibe paquetes con datos de sensores, los transmite a través del protocolo Ethernet al propio backend SigFox.
En el apartado "SigFox Support Systems" de la figura 3, se encuentra el backend de SigFox, desde donde el usuario final puede monitorizar la información enviada de sus sensores y procesarla a bases de datos o aplicaciones propias.

2.2.2 Propuesta 2

En esta propuesta se propone el uso de la tecnología LoRa para el desarrollo de una red IoT [20] [21].

LoRa se caracteriza por ser una tecnología que no ofrece una arquitectura (a diferencia de SigFox), es decir, el propio usuario debe adquirir un gateway (para permitir la recogida y distribución de la información) así como servicios backend como servidores de aplicación o de red para la puesta a punto de la información de los dispositivos IoT a una Aplicación externa.

Se caracteriza por ser una tecnología de bajo consumo energético LPWAN. Trabaja con ondas radio a distintas frecuencias, que pueden ir desde 433 MHz hasta 868 MHz (915 MHz en Estados Unidos), lo que nos permite alargar la duración de baterías/pilas de nuestros dispositivos y abarcar con un solo dispositivo grandes áreas (de diversos kilómetros).

Usa un ancho de banda de canal de 125/250 kHz, alcanzando velocidades de transferencia de 250 bps hasta 50 Mbps (en condiciones óptimas, más elevadas que SigFox) utilizando su capacidad de comunicación bidireccional. Esta comunicación se basa en una misma tasa de transferencia de datos útiles de subida o de bajada (en SigFox la de bajada quedaba penalizada).

Cuenta con seguridad basada en claves. Se definen las siguientes:

- Network Session Key: una clave única de 128 bits compartida entre el dispositivo final y el servidor de red.
- Application Session Key: una clave única de 128 bits compartida de principio a final en la capa de la aplicación.

Trabaja con un protocolo propio llamado LoRaWAN, que facilita la comunicación de los dispositivos integrantes de la red IoT con el backend de la misma, además de proporcionar niveles de seguridad extremo a extremo.

Por tanto, podemos decir que LoRa cuenta con características muy similares a SigFox, con novedades como comunicación bidireccional simétrica y tasas de transferencia más elevadas.
No obstante, a pesar de no pagar licencias por dispositivos LoRa, puesto que es gratuito, el usuario debe desplegar la arquitectura de la red, lo que llevará a realizar una mayor inversión

en tiempo. LoRa ofrece, además, una mayor personalización de la red, ya que el usuario final debe organizarla por sí mismo.

Esquema de red LoRa [21]

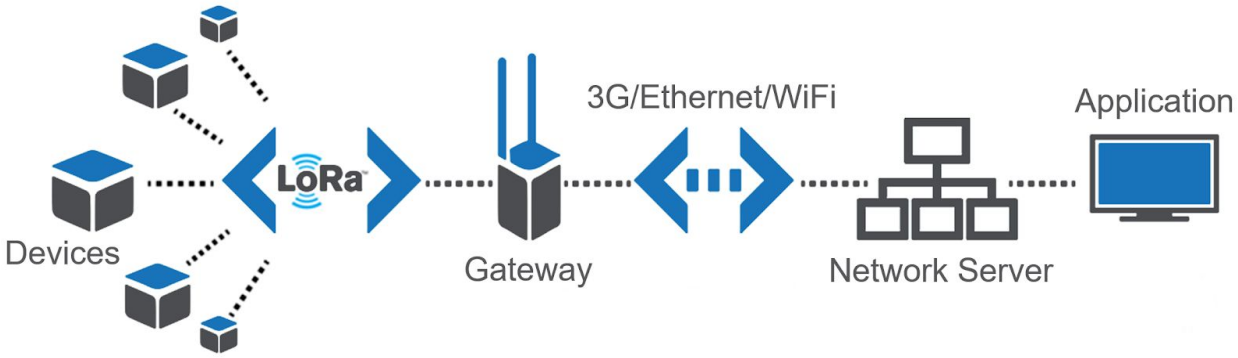


Figura 4. *Esquema de red LoRa.*

El esquema de la figura 4 es muy similar al presentado en el apartado de SigFox.

Por un lado, se pueden observar los "Devices" o nodos finales transmitiendo información mediante tecnología LoRa (conexión inalámbrica basada en antenas) a los distintos gateway trabajando a una misma frecuencia de recepción.

Por otro lado, estos gateway envían los datos recibidos de los nodos finales a un backend mediante protocolo Ethernet o WiFi/3G. El backend consta de un *Network Server* así como de un *Application Server* entre otros, dependiendo del tipo de backend a usar.

Finalmente, se pueden almacenar los datos en una base de datos propia o externa para su posterior procesado en una aplicación propia.

2.3 Justificación

En este apartado se estudia el impacto económico (esto incluye horas de desarrollo, licencias software, contratación de servidores, etc.) de las posibles propuestas presentadas en el apartado anterior.

2.3.1 Estimación de recursos

Estimación de recursos SigFox:
Para el desarrollo de una red IoT de tan solo 10 dispositivos se necesitan:

- 10 dispositivos IoT (sensores)
- 4 licencias SigFox, una para cada dispositivo,con duración de un año
- 4 controladoras de desarrollo (3 sensores por placa conforman una configuración óptima) → Arduino MKRFOX 1200

- 4 antenas para las controladoras (proporcionan conectividad LoRa con el gateway) → ISM 868/915 MHz Stand Alone 20674-0200.

Para el desarrollo de una red IoT de 10,000 dispositivos se necesitan:

- 10,000 dispositivos IoT (sensores)
- 3,334 licencias SigFox, pero de tipo enterprise (a partir de 1,000 dispositivos, SigFox proporciona esta opción, con licencias más duraderas y más ventajas
- 3,334 controladores de desarrollo (suponiendo agrupaciones de 3 sensores por placa, aunque dependería de las agrupaciones de los mismos, ya que una placa podría soportar más unidades de estos, según la configuración geográfica de los mismos podríamos poner más sensores o menos) → Arduino MKRFOX 1200
- 3,334 antenas para los controladores (proporcionan conectividad LoRa con la gateway) → ISM 868/915 MHz Stand Alone 20674-0200.

Estimación de recursos LoRa:
Para el desarrollo de una red IoT de tan solo 10 dispositivos se necesitan:

- 10 dispositivos IoT (sensores)
- 1 gateway LoRa → Laird RG186
- 4 controladores de desarrollo (3 sensores por placa conforman una configuración óptima) → Arduino MKR WAN 1300 (Conectividad LoRa)
- Loriot Network server (Community Public Network Server)
- 4 antenas para los controladores (proporcionan conectividad LoRa con la gateway) → ISM 868/915 MHz Stand Alone 20674-0200.

Para el desarrollo de una red IoT de 10,000 dispositivos se necesitan:

- 10,000 dispositivos IoT (sensores)
- 50 gateway LoRa (suponiendo unos 200 nodos por gateway) → Laird RG186
- 3,334 placas de desarrollo (suponiendo agrupaciones de 3 sensores por placa, aunque dependería de las agrupaciones de los mismos, ya que una placa podría soportar más unidades de estos, según la configuración geográfica de los mismos podríamos poner más sensores o menos) → Arduino MKR WAN 1300 (Conectividad LoRa)
- Loriot Network server (Private Network Server)
- 3,334 antenas para los controladores (proporcionan conectividad LoRa con la gateway) → ISM 868/915 MHz Stand Alone
- Servicios de la plataforma Amazon AWS, divididos en:
    - Base de datos DynamoDB:
        - 14,400,000 escrituras mensuales (48 mediciones diarias/dispositivo)
        - 100,000 lecturas mensuales (por la aplicación)
        - 25 GB capacidad de la base de datos
        - 25 GB copias de seguridad, bajo demanda y programadas
    - IoT Core:
        - Conectividad:

- ➢ 43,800 minutos de conexión mensuales de cada dispositivo (24/7)
- Mensajeria:
  - ➢ 14,400,000 mensajes de dispositivos a la plataforma mensuales
- Motor de reglas:
  - ➢ 14,400,000 reglas disparadas
  - ➢ 14,400,000 acciones ejecutadas
  - ➢ Refieren a las acciones de guarda de cada dato en la base de datos DynamoDB.

2.3.2 Impacto económico

Impacto económico SigFox:

- Para el desarrollo de una red IoT de tan solo 10 dispositivos se necesitan:

Tabla 1. *Presupuesto despliegue red Sigfox 10 dispositivos.*

Nombre recurso	Precio unitario (€)	Unidades	Subtotal (€)
Licencias SigFox	16.13	4	64.52
Arduino MKRFOX 1200	35.91	4	143.64
ISM 868/915 MHz Stand Alone 20674-0200	3.38	4	13.52
TOTAL (€)			221.68

- Para el desarrollo de una red IoT de 10,000 dispositivos se necesitan:

Tabla 2. *Presupuesto despliegue red Sigfox 10,000 dispositivos.*

Nombre recurso	Precio unitario (€)	Unidades	Subtotal (€)
Licencias SigFox (enterprise)	16.13	3,334	53,777.42
Arduino MKRFOX 1200	35.91	3,334	119,723.94
ISM 868/915 MHz Stand Alone	3.38	3,334	11,268.92
TOTAL (€)			184,770.28

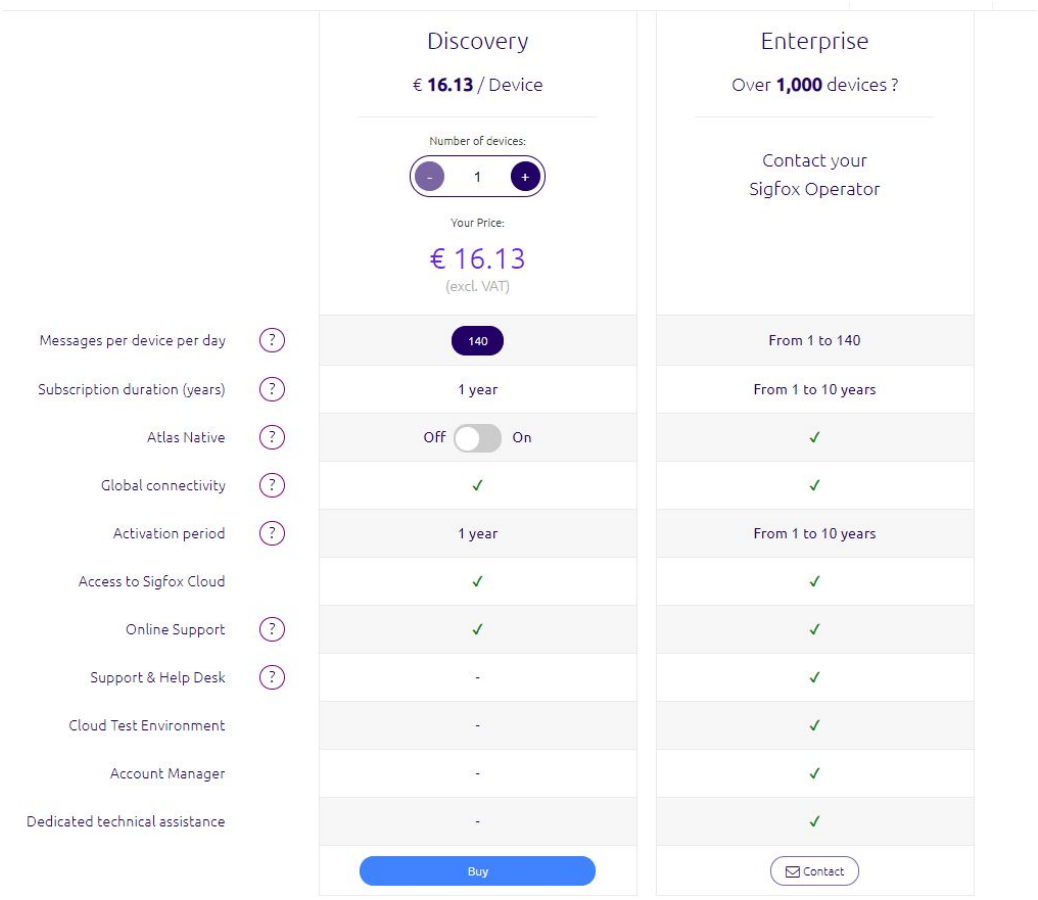


Figura 5. *Precios para el despliegue de una red Sigfox [22].*

Impacto económico LoRa:

- Para el desarrollo de una red IoT de tan solo 10 dispositivos se necesitan:

Tabla 3. *Presupuesto despliegue red LoRa 10 dispositivos.*

Nombre recurso	Precio unitario (€)	Unidades	Subtotal (€)
Laird RG186	227.45	1	227.45
Arduino MKR WAN 1300	35.05	4	140.20
Loriot Network Server (Community Public Network Server)	Gratuito	1	Gratuito
ISM 868/915 MHz Stand Alone 20674-0200	3.38	4	13.52

CloudMQTT (Free Plan)	Gratuito	1	Gratuito
Firebase (Spark Plan)	Gratuito	1	Gratuito
TOTAL (€)			381.17

- Para el desarrollo de una red IoT de 10,000 dispositivos se necesitan:

Tabla 4. *Presupuesto despliegue red LoRa 10,000 dispositivos.*

Nombre recurso	Precio unitario (€)	Unidades	Subtotal (€)
Laird RG186	227.45	50	4,847.50
Arduino MKR WAN 1300	35.05	3,334	116,856.70
Loriot Network Server (Private Network Server)	2,000.00	1	2,000.00
ISM 868/915 MHz Stand Alone	3.38	3,334	11,268.92
DynamoDB	23.89	1	23.89
IoT Core	49.00	1	49.00
TOTAL (€)			135,046.01

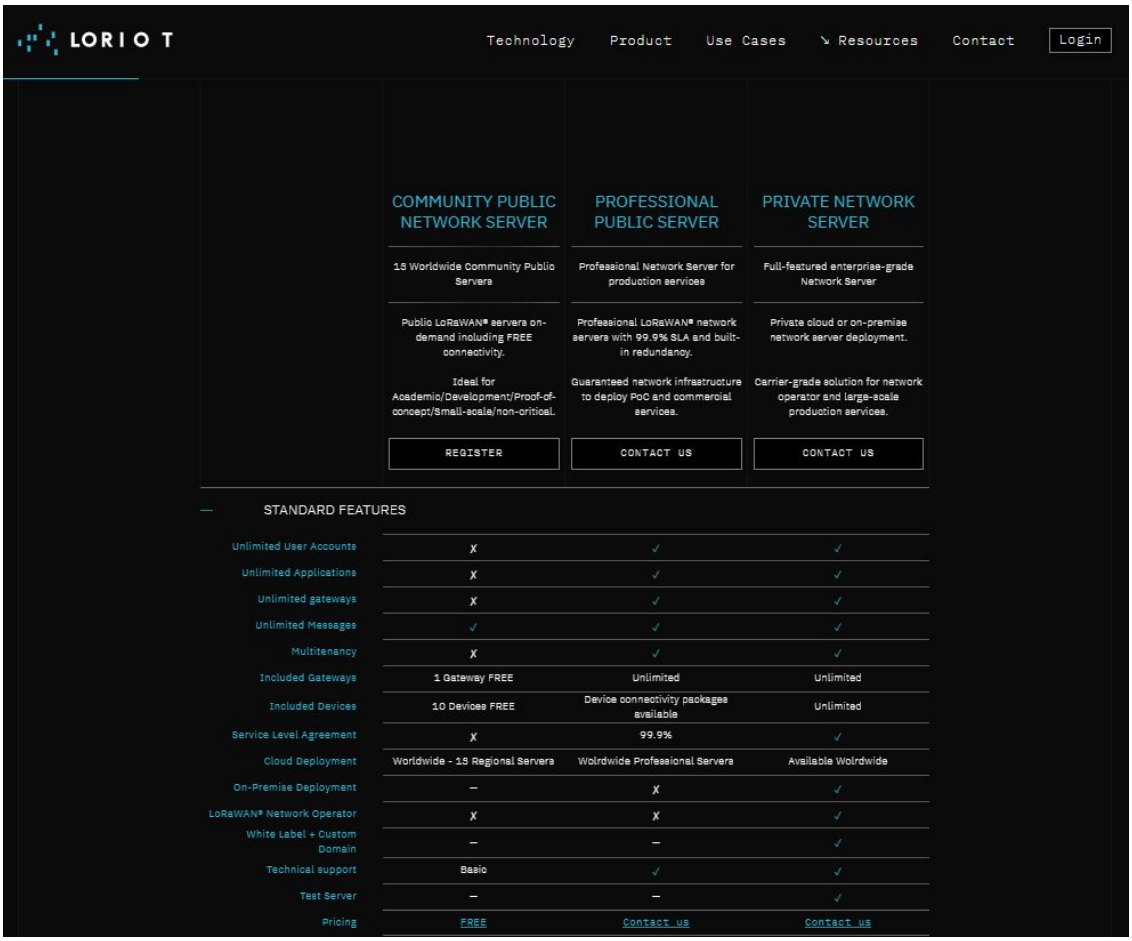


Figura 6. *Precios del backend de Loriot [3].*

2.3.3 Propuesta final

En base a las tecnologías expuestas en los apartados anteriores (SigFox y LoRa), se ha decidido implementar una red IoT con la tecnología LoRa.

A pesar de que en la estimación de recursos se ha observado que implementar una red LoRa es notablemente más complicado que implementar una red SigFox (contratación servicios Loriot para backend, compra de gateways, etc.), LoRa permite diseñar una arquitectura de red adecuada a nuestra idea de proyecto inicial, ya que es el usuario final el que la debe implementar.

En cuanto al presupuesto necesario para desarrollar una red, ambas tecnologías tienen los siguientes costes:

- Para el caso de 10 dispositivos, una red LoRa costaría **381.17€,** mientras que una red SigFox costaría **221.68€**
- Para el caso de 10,000 dispositivos una red LoRa costaría **135,046.01€,** mientras que una red SigFox costaría **184,770.28€**.

Como la mayor diferencia de presupuesto se da en el desarrollo de una red de 10,000 dispositivos, donde la red LoRa es más barata, el proyecto se llevará a cabo con LoRa, inicialmente para una red de 10 dispositivos, y finalmente una estimación teórica para una red de unos 10,000 dispositivos.

3 Implementación

En este proyecto se ha diseñado, además de la red, una aplicación móvil para acceder a los datos de los sensores almacenados en la base de datos. En la aplicación se muestran (divididas por categoría de sensor) las distintas mediciones tomadas. Se pueden también realizar operaciones especiales, como borrado de documentos de la base de datos, entre otras.

En este apartado, se definirán las partes de las que se compone la red, así como de la aplicación.

3.1. Implementación de las partes de la red

3.1.1 Configuración del gateway

Una parte importante de una red LoRa es el uso de dispositivos llamados *gateway*, que asemejan su función a la de un router convencional, es decir, encaminar paquetes a través de redes [23].
En este caso, se ha conectado dicho gateway mediante cable RJ45 a un puerto de un router convencional. De esta manera, el gateway podrá derivar los datos entrantes de las controladoras Arduino hacia el router al que se encuentra conectado. Será el propio router el que envíe los paquetes de datos hacia el backend de Loriot.

La comunicación entre las controladoras y el gateway se realiza de manera inalámbrica a través de la tecnología LoRa, a 868 MHz. Ambos dispositivos cuentan con una antena específica para esta tecnología.

A continuación, se mostrará la configuración que se le ha proporcionado al gateway a través de su cliente web para el correcto funcionamiento en la red LoRa. Al acceder a través de la dirección que ofrece el fabricante, se observa el siguiente panel.

En él, se han configurado los apartados “LoRa” y “LAN”. El apartado “WI-FI” se ha deshabilitado ya que el gateway está conectado al router mediante cable ethernet, por lo que no es necesaria la conexión inalámbrica.

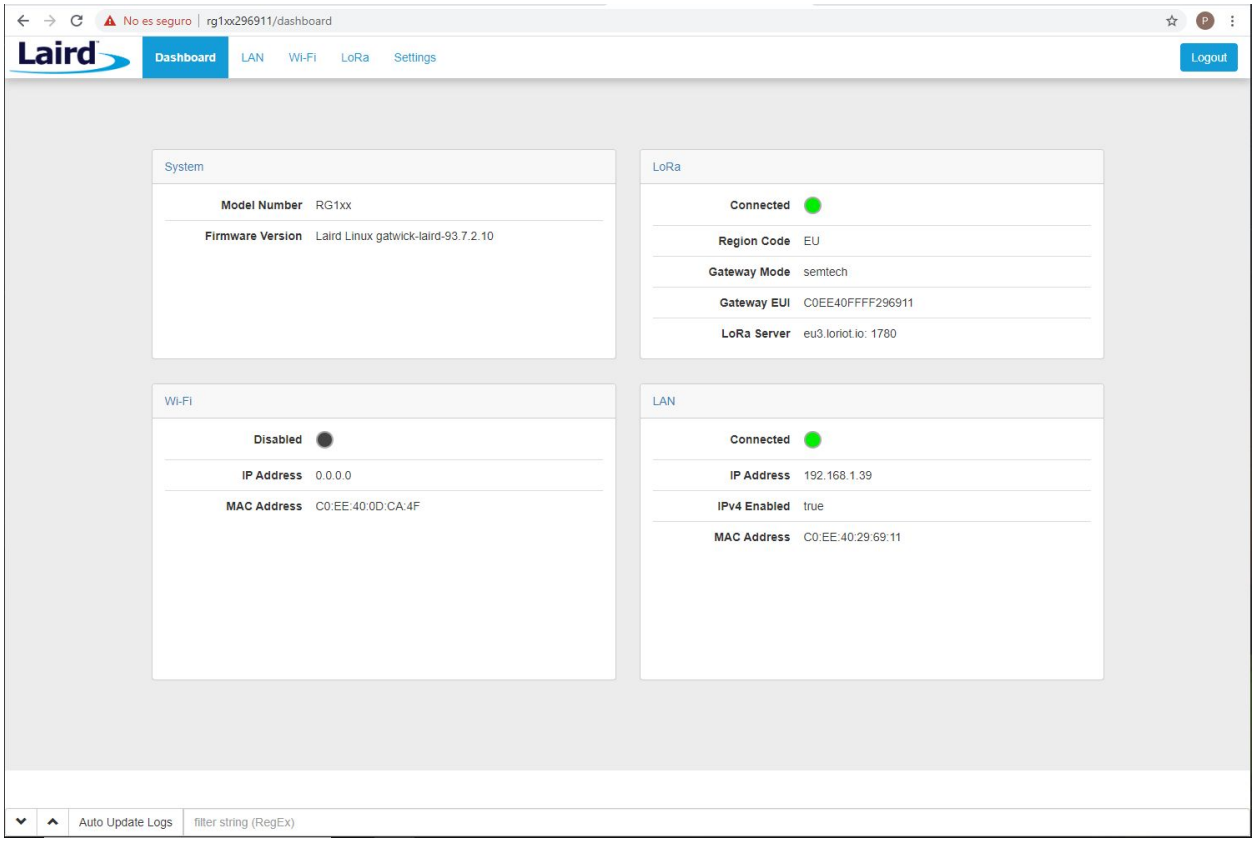


Figura 7. *Panel de control del gateway del proyecto.*

En el apartado **LAN** de la figura 8, se ha configurado la dirección IP del gateway. En este caso, la dirección IP se obtiene mediante el protocolo DHCP, encargado de dar direcciones IP a los dispositivos de una red. El propio router al que se encuentra conectado el gateway es el dispositivo que ofrece dirección IP a este último. La dirección IP es 192.168.1.39, y la máscara 255.255.255.0. El DNS es proporcionado de manera automática por el router.

No se ha configurado IPv6, por lo que las opciones de IPv6 se han dejado por defecto.

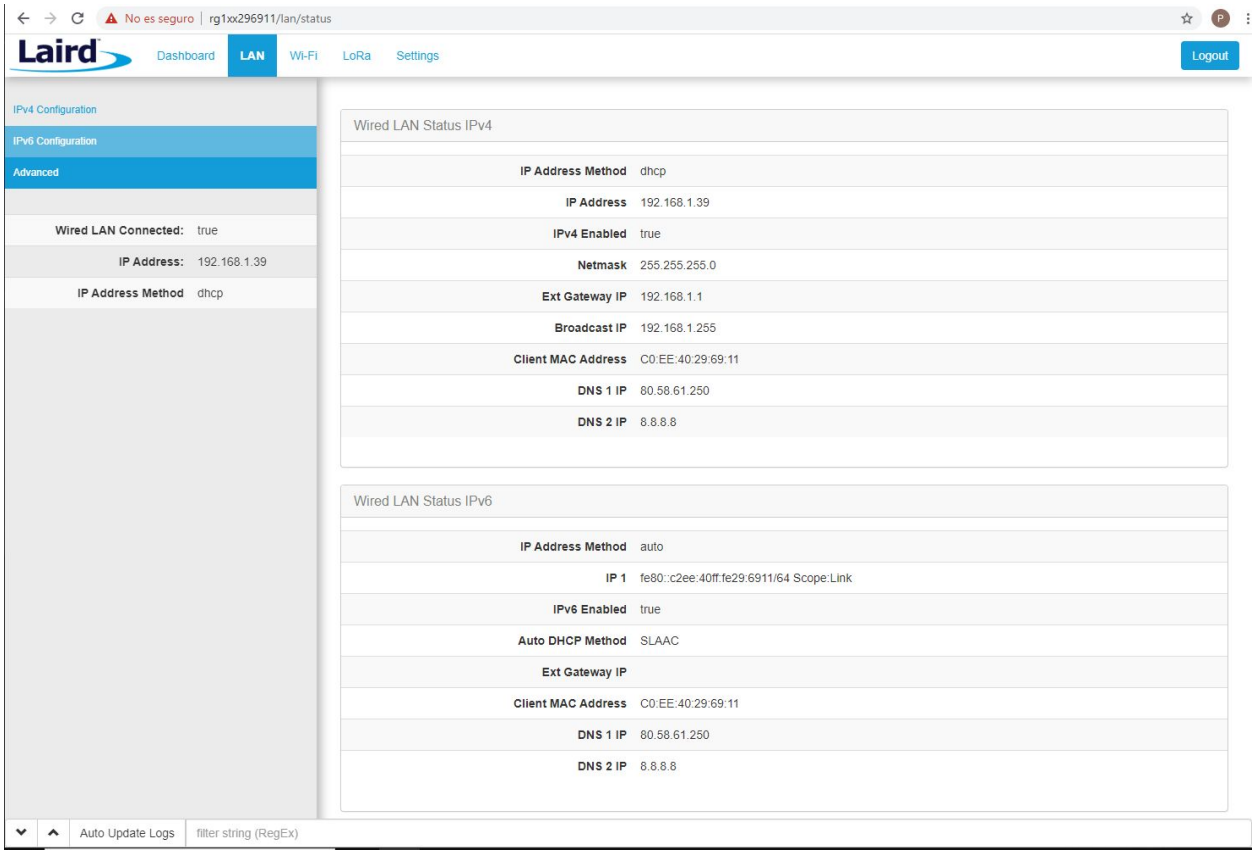


Figura 8. *Apartado LAN del panel de control del gateway del proyecto.*

En el apartado **LoRa** de la figura 9, se ha configurado la conexión del gateway con el backend de Loriot.

El propio gateway ofrece en el apartado “Presets” diversas conexiones ya configuradas a distintos backends (entre ellos Loriot) para la correcta conexión del gateway. No obstante, se ha ignorado esta opción debido a unos cambios en el firmware del gateway, que cambiaron la manera de conectar el gateway a Loriot.
Por ello, en el apartado "Forwarder”, se ha realizado la configuración manual del servidor Loriot que gestionará los paquetes enviados por el propio gateway de los dispositivos de la red local.

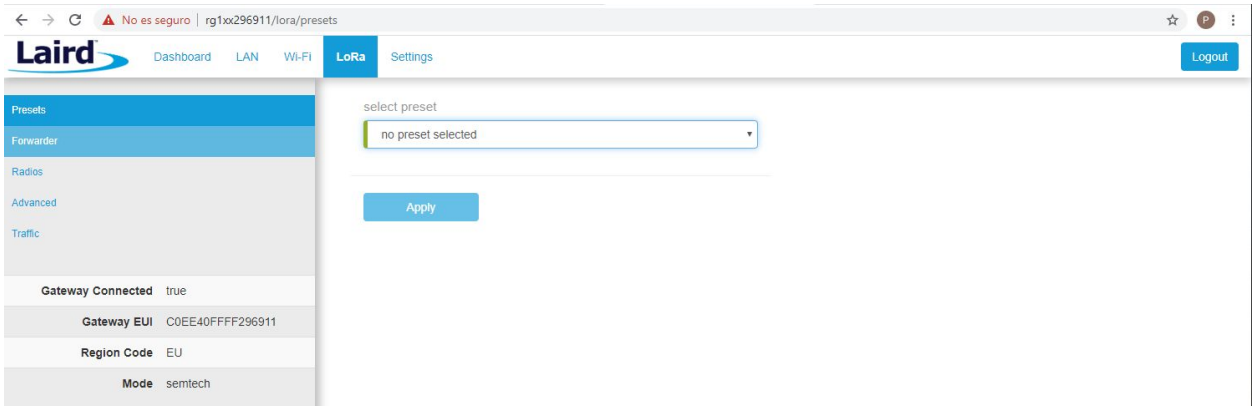


Figura 9. *Apartado LoRa del panel de control del gateway del proyecto.*

El modo especificado es **Semtech Forwarder**, que es uno de los distintos modos soportados por el backend Loriot. En este modo tan solo se han especificado la dirección del servidor Loriot al que se conectará el gateway y el puerto por el que se realizará la comunicación. La dirección del servidor es **eu3.loriot.io,** que es el servidor europeo de Loriot. El puerto es el 1780.

Todos los datos de conexión se especifican en la propia página de Loriot.
Como se puede observar en la figura 10, el estado del gateway es conectado, por lo que se ha vinculado de manera satisfactoria en el backend. También se muestra el identificador del dispositivo o EUI, así como la región en la que estamos trabajando, que en este caso es EU (Europa).

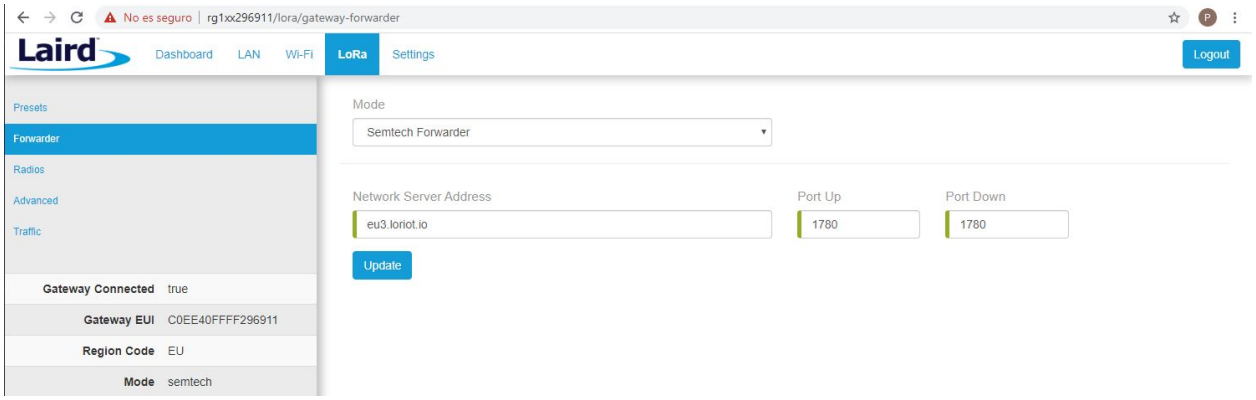


Figura 10. *Apartado LoRa del panel de control del gateway del proyecto.*

Los demás apartados no han sido relevantes a la hora de configurar el gateway (a parte de opciones como el cambio de nombre de usuario y contraseña del propio gateway para securizar el acceso al mismo a través de navegador web).

A partir de este punto, ya se ha configurado el gateway para trabajar en la red LoRa creada por el propio backend de Loriot, a partir de la vinculación del gateway al mismo. Para que las controladoras Arduino puedan enviar datos al gateway, no se precisa ninguna configuración

previa, ya que el gateway recibirá todos los paquetes que sean transmitidos en su frecuencia de escucha, es decir, 868 MHz.

3.1.2 Configuración del backend

En este apartado se explicará la configuración del backend, que recibirá los paquetes que le envíe el gateway LoRa a través del router. Este backend se constituye, como se ha explicado en apartados anteriores, de un servidor de red, así como de un servidor de aplicación.

En primer lugar, se ha configurado la parte de red. Para ello, se ha creado una red con nombre "TFG", que se puede apreciar en la siguiente figura 11.

Loriot asigna por defecto un identificador a cada red o aplicación creada. En este caso, este identificador es **A0000114**. Además, se han configurado parámetros como la localización de la red, que en este caso es Alcoy, concretamente en la zona con código postal 03803. No obstante, la localización no será un factor determinante en el funcionamiento de la red, es meramente informativo.

Figura 11. *Apartado de red del panel de control del backend Loriot.*

En la figura 12, se puede observar el registro del gateway LoRa dentro de la red anteriormente creada. A pesar de haber conectado el gateway anteriormente al backend, hay que realizar dicha conexión en ambos lados.

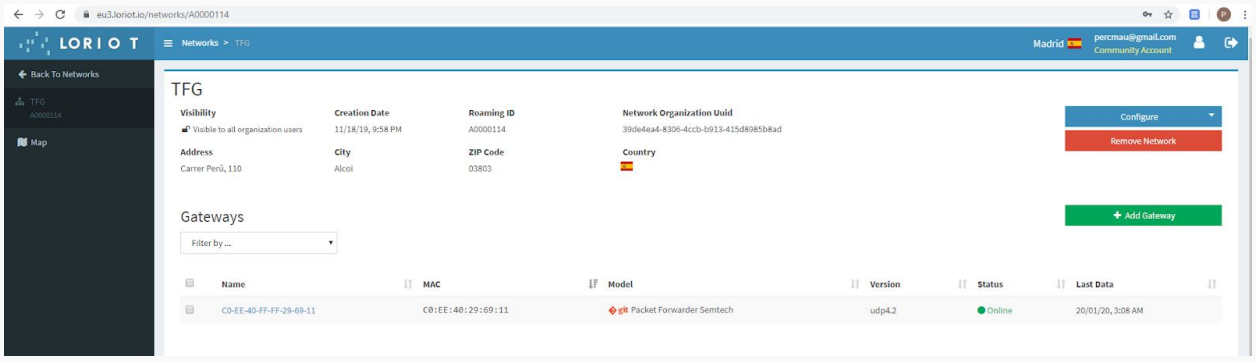


Figura 12. *Apartado de red del panel de control del backend Loriot.*

Para el registro del gateway se han seguido unos pasos distintos a como se especificaba en el propio manual del fabricante por una actualización del firmware del gateway. En esta versión del firmware, el fabricante dejaba sin soporte de conexión con el backend Loriot.
Después de contactar con ambas partes, Loriot nos ofreció la solución de registrar el gateway como Packet Forwarder Semtech, y no como Laird RG1, que es el propio modelo del gateway.

Figuras 13 y 14. *Apartado de registro del gateway del panel de control del backend Loriot.*

Como se ha especificado en el punto **3.1.1. Configuración del Gateway**, se ha configurado como forwarder principal en el gateway, el forwarder de Semtech, por lo que al seleccionar el mismo forwarder en Loriot, la comunicación entre ambas partes será posible.

Por otra parte, a la hora de registrar el propio gateway, se debe de especificar, entre otros parámetros, el *eth0 MAC address,* que es uno de los distintos identificadores que posee el gateway, y que son únicos en cada gateway. En este caso, la mac requerida es "C0:EE:40:29:69:11", y puede ser obtenida en la parte inferior del gateway, especificada por el fabricante.

Además, se pide que se especifique la localización del gateway, que será, en este caso, la misma que se ha especificado en la red, para así poder geolocalizar desde el backend nuestros dispositivos de red.

Una vez registrado el gateway, podemos observar en la figura 15, que el propio sistema ha especificado de manera automática el identificador del dispositivo o DevEUI (en la imagen es el campo **Name**). Este identificador es uno de los tres que se pueden observar en la parte inferior del gateway.

Dentro del registro del gateway, Loriot ofrece distintas opciones de configuración y monitorización del propio gateway, que son útiles a la hora de enviar y recibir paquetes. Se muestra en la figura 15.

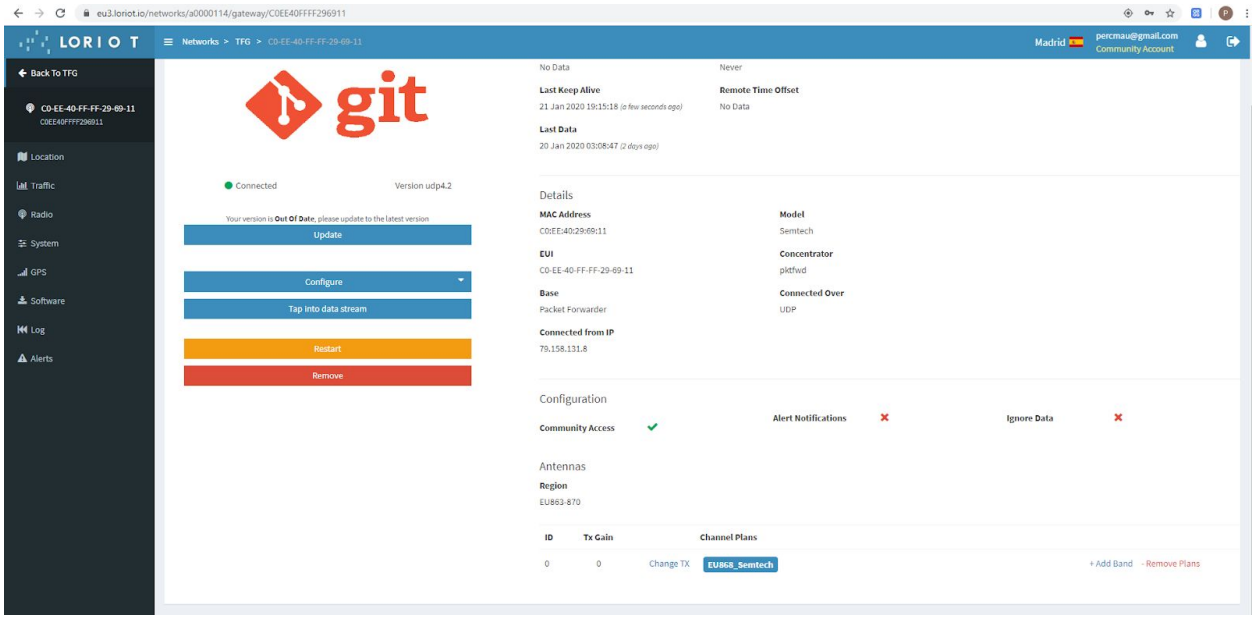


Figura 15. *Apartado del gateway del panel de control del backend Loriot.*

Estas opciones serán comentadas en apartados posteriores, donde se enviarán mensajes simples como "Hola Mundo" hacia el backend para verificar la correcta relación entre el gateway y el backend.

Una vez configurada la parte de red, se configura la parte de aplicación, también desde Loriot.

En primer lugar, se crea una aplicación, en este caso con mismo nombre que la red "TFG".

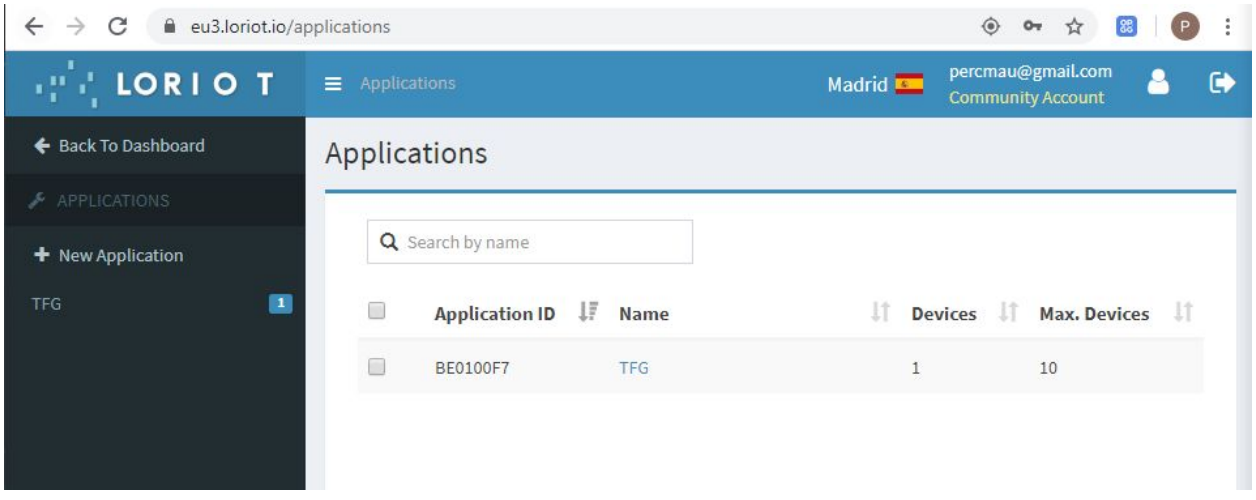


Figura 16. *Apartado de aplicación del panel de control del backend Loriot.*

Una vez creada la aplicación, accedemos a la misma para registrar los dispositivos que enviarán información. En este caso, se añade un dispositivo con los datos de la controladora MKR WAN 1300. Seleccionando la opción "Enroll Device" se puede registrar un dispositivo.

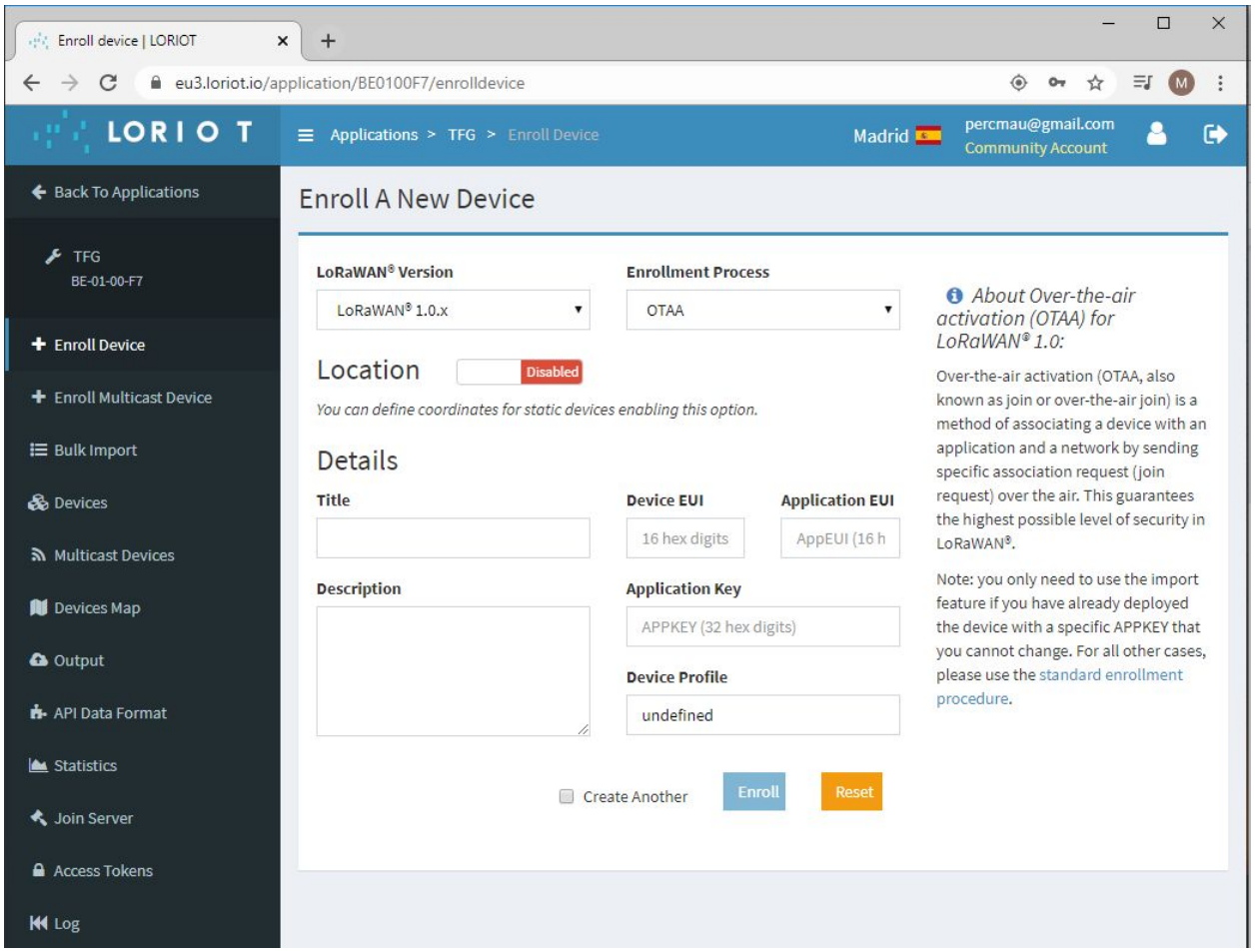


Figura 17. *Apartado de registro de un dispositivo del panel de control del backend Loriot.*

En el apartado "Enrollment Process" se define la manera de crear los identificadores necesarios. Pueden ser introducidos manualmente, o dejar que todos, o algunos de ellos, sean creados automáticamente por el sistema.
En este caso, se selecciona "Generate all parameters except DevEUI", es decir, se introduce el identificador de la controladora de manera manual, y se generan de manera automática los identificadores de la aplicación anteriormente creada.
El identificador de la controladora puede ser obtenido mediante un código básico en Arduino que, al ser ejecutado sobre una controladora, devuelve el identificador que ha establecido el fabricante.

De esta manera, el backend está listo para recibir paquetes de datos de los componentes que forman la red. Solamente se ha necesitado realizar la creación de una red LoRa y una

aplicación con los respectivos dispositivos que envían datos a la red, en este caso, las controladoras Arduino.

3.1.3 Configuración de la controladora

El código usado corresponde al código del apartado **8.1 Códigos - Script MKR WAN 1300 [24].**

La controladora MKR WAN 1300 es una controladora Arduino con conectividad LoRa que nos permite el envío de la información de los sensores hacia el gateway [24].

La configuración y uso de la misma se realizarán a través del software gratuito que Arduino proporciona, el IDE de Arduino.

En todos los códigos que se ejecutarán sobre la controladora, se necesitará configurar una serie de identificadores ofrecidos por el propio backend. Estos identificadores serán usados por el script para poder determinar a qué backend deberán de ser enviados los datos generados. Se definen en un fichero llamado **arduino_secrets.h.** El fichero tiene la forma mostrada en la figura 18.

arduino_secrets.h

```
1  // Replace with keys obtained from TheThingsNetwork console
2  #define SECRET_APP_EUI "xxxxxxxxxxxxx"
3  #define SECRET_APP_KEY "yyyyyyyyyyyyyyyyyyyyyyyy"
4
```

Figura 18. *Credenciales del script del módulo MKR WAN 1300.*

A continuación, se mostrarán los parámetros que nos ofrece el propio backend.

En primer lugar, el identificador APP_EUI definido por la variable SECRET_APP_EUI del código, puede ser encontrado en Loriot en la ruta **Applications > TFG > Devices**, accediendo al dispositivo registrado en cuestión. En la figura 19, se muestran los siguientes parámetros.

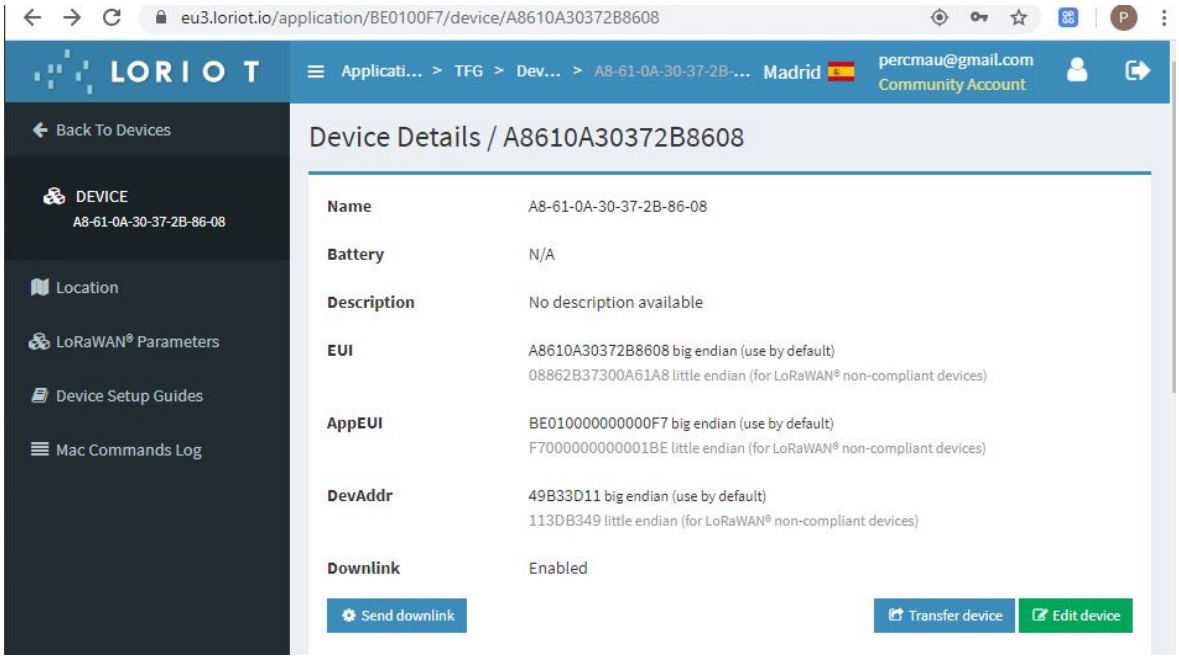


Figura 19. *Apartado de control de un dispositivo del panel de control del backend Loriot.*

Para saber el identificador APP_KEY definido por la variable SECRET_APP_KEY del código hay que desplazarse a la ruta de la figura 20, y acceder al apartado LoraWAN Parameters. Se nos muestran distintos identificadores, pero el que se requiere es AppKey.

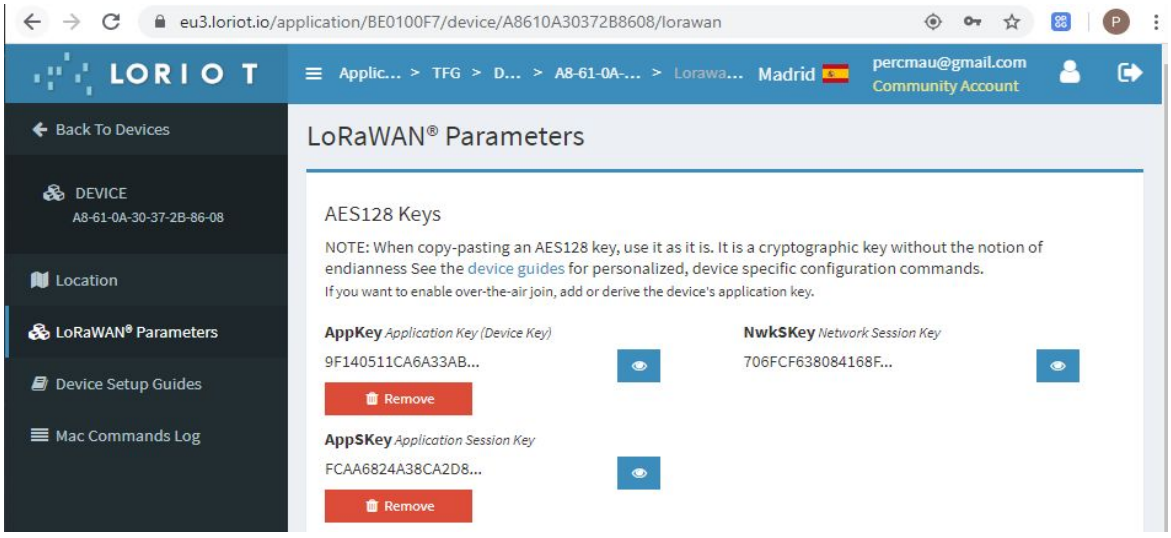


Figura 20. *Apartado de control de un dispositivo del panel de control del backend Loriot.*

Una vez conocidos los datos necesarios, e introducidos en el fichero **arduino_secrets.h**, se procederá a realizar las pruebas pertinentes en el apartado de pruebas. El fichero queda como se observa en la figura 21.

arduino_secrets.h

```
1  // Replace with keys obtained from TheThingsNetwork console
2  #define SECRET_APP_EUI "BE010000000000F7"
3  #define SECRET_APP_KEY "9F140511CA6A33ABED0B00D2FD4F3A7F"
4
```

Figura 21. *Credenciales del script del módulo MKR WAN 1300.*

Otra opción importante en el código es definir la frecuencia a la que se van a emitir los paquetes de datos, que en este caso, coincidirá con la frecuencia a la que emite la propia antena de la controladora. En nuestro caso, definimos **EU868** (Europa a 868 MHz).

De esta manera, la controladora Arduino queda totalmente funcional, restando posibles configuraciones en el propio código, según el que se quiera emplear.

3.1.4 Configuración del broker MQTT

Para la extracción de los paquetes del backend de Loriot es necesario un bróker de mensajería, que no es más que un intermediario encargado de traducir los mensajes de un sistema origen a uno destino [25].

En este proyecto, se ha empleado el protocolo MQTT, que es un protocolo ligero de mensajería P2P que utiliza el modelo de comunicación Publish-Subscribe. Se caracteriza por emplearse en entornos con mensajes de poco peso, conexiones lejanas y restricciones en el ancho de banda disponible.

Entre las distintas ventajas que ofrece MQTT, destacamos la flexibilidad en la comunicación entre los distintos participantes de la misma, gracias a las colas de mensajes. Las colas son las encargadas de almacenar los mensajes que los nodos publiquen, con lo cual, no es necesario los emisores y receptores interactuen con la cola a la vez. Los receptores de los mensajes pueden acceder a la cola en cualquier momento para recibir los mensajes esperados, siempre que no se elimine el mensaje por exceso de tiempo en cola.

Figura 22. *Esquema del protocolo de comunicación publish-subscribe [13].*

En la figura 22, se define el modelo de mensajería publish-subscribe usado por el propio broker.
Un nodo "Producer" o emisor publica datos en un tópico, que no es más que una forma de agrupar los mensajes de una comunicación entre un emisor y uno o varios receptores. En el

esquema, el backend Loriot sería el encargado de publicar los paquetes con la información de los sensores en el broker de CloudMQTT.
Un nodo "Consumer" o receptor es aquel que se suscribe a un tópico para recibir los datos publicados por un emisor al mismo. En este caso, pueden haber tantos receptores como soporte la infraestructura. Un nodo receptor también puede publicar datos sobre un tópico, para que otros nodos suscritos los reciban. En este caso, el receptor de los datos publicados en el broker es un código de Python, que se encargará de recibir y almacenar los datos de los sensores en una base de datos.

En cuanto a la configuración del broker, se ha creado una instancia con el nombre "TFG". Esta instancia es gratuita, ya que dentro de la página web se ofrecen distintos planes, donde cada uno ofrece distintos recursos al usuario. De todos ellos, uno es gratuito, y cumple con los requisitos de nuestra red. Ofrece:

- 5 conexiones máximas al broker
- Reglas ACL
- Transferencias de datos de 10 Kbit/s como máximo.

Para este proyecto, estos recursos son más que suficientes.

A la hora de crear la instancia, tan solo se deberá seleccionar "Crear nueva instancia" y otorgar un nombre a la instancia, en este caso "TFG". También se deberá seleccionar el servidor en el que se quiera alojar la instancia, que en este caso será Europa, por proximidad.

Acto seguido, la página web devolverá la instancia creada, con los datos de conexión a la misma autogenerados, a punto de ser usada. El resultado es el de la figura 23.

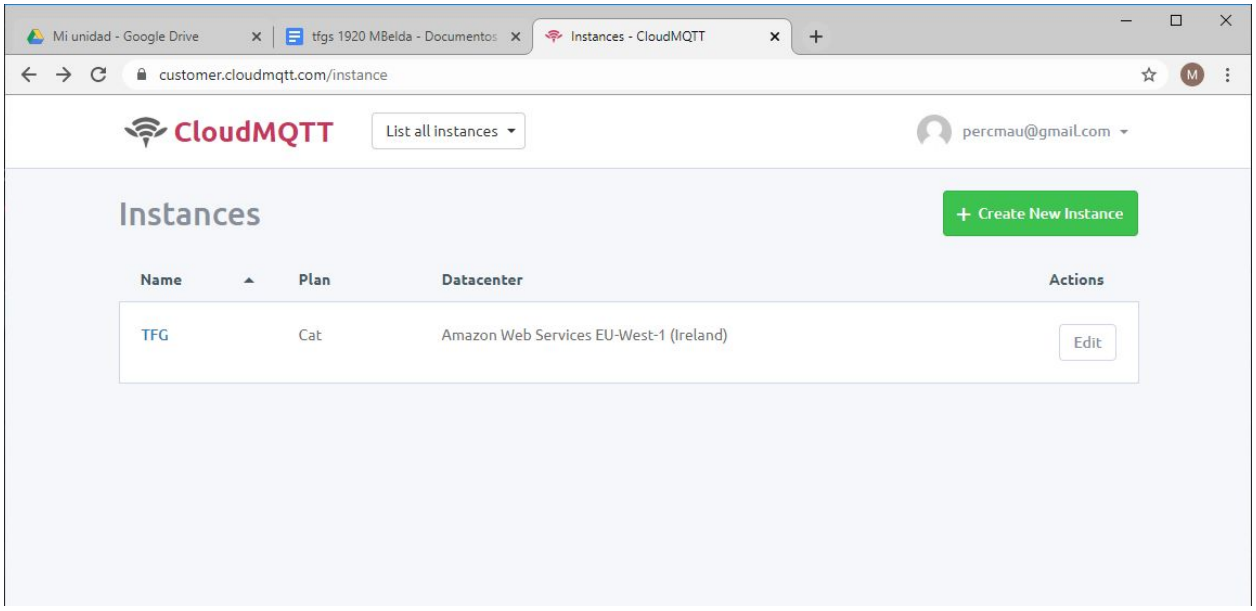


Figura 23. *Panel de control del broker cloudmqtt.*

Como se observa en la figura 24, dentro de la instancia, se encuentran los datos de la misma para la correcta conexión con el publicador y los posibles suscriptores. Entre estos, podemos destacar la dirección del servidor que se nos ha asignado, un usuario y una clave para el acceso al mismo, así como distintos puertos de conexión. Los puertos de conexión se distinguen por [13]:

- Port: conexión directa mediante protocolo MQTT
- SSL Port: conexión mediante protocolo MQTT y SSL (protocolo criptográfico que proporciona seguridad a la comunicación a nivel de red)
- Websockets Port (TLS only): conexión mediante protocolo MQTT, Websockets y TLS que permite la conexión directa al broker desde un portal web usando javascript (mejora de seguridad respecto a SSL).

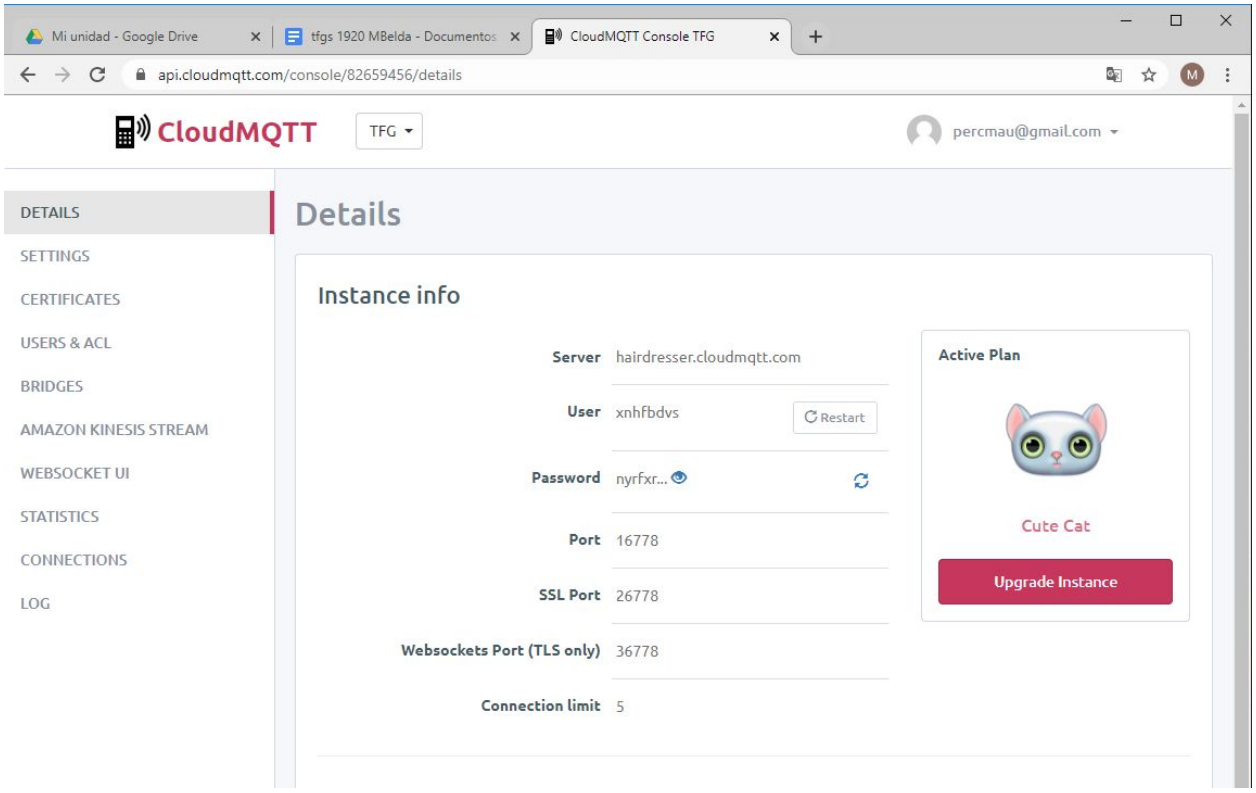


Figura 24. *Panel de control del broker cloudmqtt.*

En este punto, el backend estaría listo para ser usado, tan solo falta vincularlo al backend y, por otra parte, vincularlo a un script para la obtención de los datos.

3.1.4.1 Conexión backend-broker

Para que el broker de mensajería reciba los mensajes del backend Loriot, se debe configurar en el propio backend una salida de datos. Dentro de la aplicación creada para la recepción de

los datos de los sensores, se configura una salida mediante protocolo MQTT. En la figura 25 se detallan los datos de la conexión al broker.

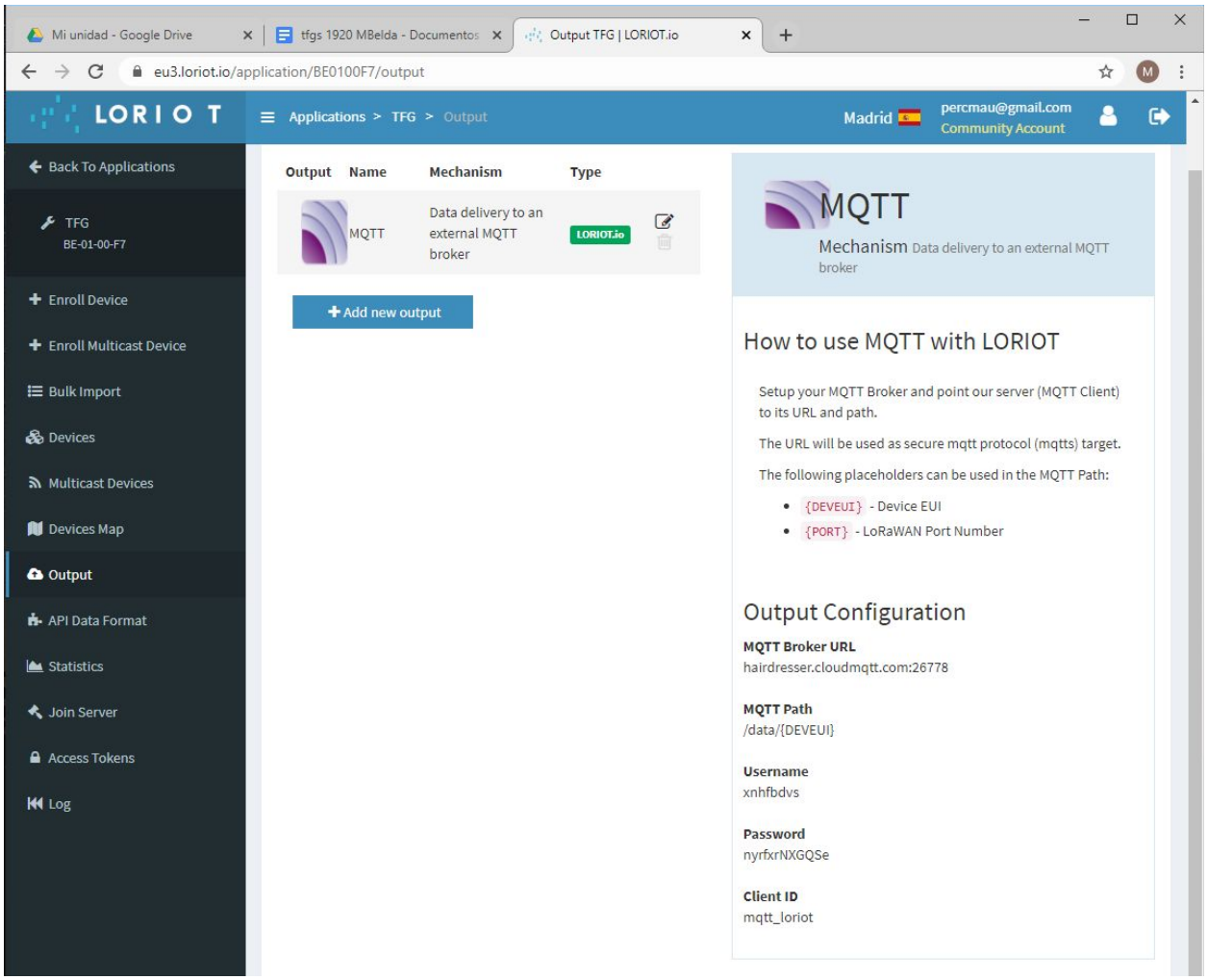


Figura 25. *Salidas de datos del backend de Loriot.*

En el apartado del servidor del broker se detalla la URL del mismo así como el puerto por el cual se realizará la conexión. En este caso se ha elegido el puerto SSL a modo de securizar la red.

El path del protocolo MQTT es /data/{DEVEUI}, por lo que distribuiremos el tráfico de cada controladora Arduino en distintos directorios. Este path es lo que se conoce como tópico en el modelo publish-subscribe, es decir, la cola de mensajes donde estos serán publicados.

En los apartados "Username" y "Password" se detallan el usuario y la contraseña que ofrece el broker para la correcta conexión al servidor.

El apartado “Client_ID” puede ser autogenerado por el propio servidor, aunque en este caso se le ha asignado un valor cualquiera.
Este identificador se asocia a los publicadores y suscriptores de mensajes. Cuando un cliente se suscribe a un tópico, la suscripción se asocia al identificador del propio cliente, y no al usuario ofrecido por el servidor. Cuando se publican mensajes en un tópico, el broker itera a través de todas las suscripciones para tratar de identificar si hay alguna conexión activa con el identificador de dicho cliente. Si encuentra una conexión, el mensaje se publica al dispositivo activo. En caso contrario, el mensaje es retenido.

3.1.4.2 Conexión broker-Python

El código usado corresponde al código del apartado **8.1 Códigos - Código Python [28].**

En este apartado se va a detallar y explicar el código empleado para sustraer los datos del broker MQTT. Se ha usado el IDE Pycharm y el lenguaje Python.
Se han implementado, también, las librerías necesarias para el correcto funcionamiento del sistema. A continuación se detallan las imagenes del código por partes [26].

Se realizan las importaciones de las librerías necesarias. Se usa la librería del protocolo MQTT para la conexión con el broker. También se hace uso de la librería de Firebase para un posterior almacenado de los datos.
Se declaran tres variables globales, que son “decodificado”, “json_data” y “idcontroladora”. Serán explicadas posteriormente.

```
import paho.mqtt.client as mqttClient
import time
from datetime import datetime
import json
import firebase_admin
from firebase_admin import credentials, firestore

decodificado = ""
json_data = ""
idcontroladora = ""
```

Figura 26. *Variables y librerías del script de Python.*

Método empleado para saber si la conexión del script hacia el broker ha sido satisfactoria. El parámetro “rc” pasado al método especificará, según su valor, si la conexión se ha realizado correctamente (devuelve un 0) o, por el lado contrario, si la conexión ha sido rechazada (valores comprendidos entre 1-5).

El método se apoya en una variable global llamada “Connected” para determinar en todo momento si la conexión sigue siendo correcta. También se muestran salidas por pantalla del estado de la conexión.

```
def on_connect(client, userdata, flags, rc):
    if rc == 0:

        print("Conexión al broker exitosa")

        global Connected  # Use global variable
        Connected = True  # Signal connection

    else:

        print("Conexión fallida")
```

Figura 27. *Método on_connect() del script de Python.*

Método empleado para interceptar mensajes publicados por el broker a un tópico al que estemos suscritos. En el, se dota de valor a las variables globales anteriormente mencionadas.

En primer lugar, se guarda en la variable global “json_data” el paquete recibido.

Luego, se hace un cast y se guarda en la variable global “parsed_json” el contenido de la variable “json_data”. Esto se debe a que la cabecera y la carga útil del paquete se reciben en formato JSON.

En la condición, se realiza una selección de paquetes y de su contenido. Cuando el broker CloudMQTT publica un mensaje recibido, publica el mensaje en tres paquetes distintos, que son publicados en el tópico. Se dividen de la siguiente manera, según el campo “cmd”:

- RX: paquete recibido por el broker. Contiene cabecera y la carga útil
- TX: paquete que va a ser enviado hacia el script de python (hacia los suscriptores). Es similar al paquete RX, con diferencias en la cabecera
- Otro: paquete sin carga útil.

En este caso, solamente escogemos, para evitar duplicados, el paquete recibido por el broker, aunque se podría haber elegido el paquete enviado por el broker. Ambos contienen la misma carga útil. Se guarda en la variable local “bd” la carga útil del paquete, a modo de indexación en vectores.

Dado que la carga útil se muestra en hexadecimal, se realiza una conversión a texto plano mediante el método predefinido decode().
Luego, se muestra el contenido de la carga útil, en este caso la temperatura enviada por el sensor.

Por último, se guarda en la variable “idcontroladora” el identificador o “EUI” de la controladora Arduino que ha enviado el mensaje.

```
def on_message(client, userdata, message):
    global json_data
    global decodificado
    global idcontroladora
    json_data = message.payload.decode()
    parsed_json = (json.loads(json_data))
    if parsed_json["cmd"] == "rx":
        bd = parsed_json["data"]
        decodificado = float(bytes.fromhex(bd).decode('utf-8'))
        print("Temperatura recibida")
        idcontroladora = parsed_json["EUI"]
```

Figura 28. *Método on_message() del script de Python.*

En este apartado se inicializa la variable global “Connected” a falso, ya que no se ha iniciado aún la conexión.
Además, se declaran los parámetros que definen la conexión, así como la dirección y puerto del broker, el usuario y la contraseña. Estos parámetros son obtenidos del propio servidor del broker.

```
Connected = False  # variable global para el estado de la conexión

broker_address = "hairdresser.cloudmqtt.com"  # Dirección del broker
port = 16778  # Puerto del broker
user = "xnhfbdvs"  # Usuario de la conexión
password = "nyrfxrNXGQSe"  # Contraseña de la conexión
```

Figura 29. *Variables de conexión al broker.*

A continuación, se crea un cliente o instancia del protocolo MQTT, que en este caso hemos llamado “prueba”. A este objeto cliente se le asignan los parámetros de usuario y contraseña definidos en el apartado anterior. De esta manera, se ha creado un objeto cliente con ciertos parámetros necesarios para la conexión.

Acto seguido, se pasan los métodos definidos anteriormente “on_connect” y “on_message” a sus respectivos métodos predefinidos por el protocolo MQTT. Estos métodos predefinidos son llamados “Callbacks”, que no son más que funciones llamadas en respuesta a un determinado evento. Explicamos la función de ambas:

- On_connect: es llamada cuando se detecta una conexión broker-Python. Llama al método que hemos definido como on_connect()

- On_message: es llamada cuando se recibe un mensaje. Llama al método que hemos definido como on_message()

Ahora, se pasan los parámetros de conexión, tanto los almacenados en el objeto cliente como los almacenados en variables.
También se crea un bucle, el cual mantendrá la conexión activa en todo momento.
Se establece un bucle que mantendrá el script de Python en ejecución en caso de que la conexión se pierda en algún momento.

Por último, se suscribe al objeto cliente al tópico donde se publicarán los datos de los sensores, para así obtener nuevos mensajes publicados al mismo. El tópico es un directorio, en este caso, su path es /data/{DEVEUI} (DEVEUI es el identificador de la controladora Arduino que recoge y envía los datos de sus sensores).

```
client = mqttClient.Client("prueba")  # create new instance
client.username_pw_set(user, password=password)  # set username and password
client.on_connect = on_connect  # attach function to callback
client.on_message = on_message  # attach function to callback

client.connect(broker_address, port=port)  # connect to broker

client.loop_start()  # start the loop

while Connected != True:  # Wait for connection
    time.sleep(0.1)

client.subscribe("/data/A8610A30372B8608")
```

Figura 30. *Inicio de la conexión al broker.*

3.1.5 Configuración de la conexión Python-Base de datos

En este apartado se va a explicar el código empleado a la hora de conectar nuestro código de Python a la base de datos empleada. En este caso se ha usado la base de datos Cloud Firestore de Firebase.

En la figura 31, se muestra la conexión a la base de datos. Se ha utilizado un fichero autogenerado por Firebase con nuestros datos de conexión a nuestra base de datos. Este fichero es “ServiceAccountKey.json”, y contiene distintas keys, el identificador del proyecto en el que se incluye la base de datos, URLs de conexión, etc.

Con el método Certificate(), se obtienen las credenciales del fichero json y los almacenamos en la variable cred. A continuación, con initialize_app(), se inicializa el proyecto firebase ligado a

las anteriores credenciales. Por último, se crea un cliente para la conexión a la base de datos Cloud Firestore.

```
cred = credentials.Certificate('./ServiceAccountKey.json')
firebase_admin.initialize_app(cred)
db = firestore.client()
```

Figura 31. *Inicio de la conexión a la base de datos.*

A continuación se detalla el código empleado para guardar los datos recibidos en la base de datos. Se usa un bucle indefinido que estará a la espera de que se reciban datos.
En la condición del if se especifica que cuando la variable "json_data" tenga algún valor, se procederá a procesar y guardar dicho valor en la base de datos. Esta variable es la que almacena los datos de cada sensor al ser recibidos.

Como en la base de datos se guarda la fecha y hora en que se recibe el mensaje, utilizamos la función predefinida "datetime.now()". Esto sucederá cada vez que se reciba un mensaje nuevo. La hora se guarda en formato H:M:S, mientras que la fecha se guarda en formato D/M/A para que resulte legible por un usuario que manipule la base de datos.

Para realizar la inserción de los datos en firebase, especificamos sobre qué colección vamos a realizar la inserción, con db.collection(u'temperatura'), donde "temperatura" es el nombre de la colección en cuestión. Con document() especificamos que el id de cada documento va a ser autogenerado por la propia plataforma de la base de datos, siendo una cadena de letras y números aleatoria [27].

Con set() se especifican los "campos" del documento sobre los que se van a escribir los datos. Cada documento contará con una fecha, el nombre de la controladora que ha enviado la medición y el valor de la medición.

Finalmente, se reinicia el valor de la variable "json_data" para evitar realizar escrituras repetitivas en la base de datos.

```
try:
    while True:
        time.sleep(1)

        if json_data != "":
            fecha = datetime.now()
            fecha_actual = fecha.strftime("%d/%m/%Y %H:%M:%S")

            doc_ref = db.collection(u'temperatura').document()
            doc_ref.set({
                u'fecha': fecha_actual,
                u'nombrecontroladora': idcontroladora,
                u'valor': decodificado
            })
            print("Los datos se han introducido correctamente en la base de datos")

            json_data = ""
```

Figura 32. *Inserción de datos a la base de datos.*

En último lugar, se ha definido una excepción para el "try" anterior. La excepción "KeyboardInterrupt" se dará cuando el usuario pulse la combinación de teclas "ctrl-c". Esto servirá para dar por finalizada la conexión, realizando un "disconnect()" del objeto client y terminando el bucle que mantiene la conexión abierta.

```
except KeyboardInterrupt:
    client.disconnect()
    client.loop_stop()
```

Figura 33. *Fin de la conexión del cliente.*

3.1.6 Diseño de la base de datos

Antes que nada, se va a explicar la estructura que sigue Firebase para almacenar los datos en Cloud Firestore. A diferencia de bases de datos como MySQL, Firebase nombra de manera distinta las tablas y los registros. Una tabla de MySQL equivale a una colección de Firebase, mientras que un registro de MySQL equivale a un documento de Firebase. Los campos de Firebase refieren a los atributos en MySQL [29].

Esta base de datos es no relacional, por lo que no existen relaciones entre las distintas colecciones de la misma. Además, las colecciones no tienen un esquema a seguir por sus documentos, ya que cada documento de una misma colección puede tener unos campos en específico.

Cada colección almacena la información de un tipo determinado de sensores. En el caso de los sensores de temperatura, la información de los mismos será almacenada en una colección llamada “temperatura”.

En este caso, no hay claves primarias, pero sí se pueden añadir reglas para el control de la duplicidad, entre otros factores.

Cada tabla almacena la fecha de guardado de los datos, el valor de la medición, y la controladora que envía los datos de la medición. La fecha y el nombre de la controladora se almacenan como cadenas de caracteres, mientras que el valor se almacena en formato número.

Un ejemplo de la colección “temperatura” (análogo a las demás colecciones) y de sus campos es el que se muestra en la figura 34.

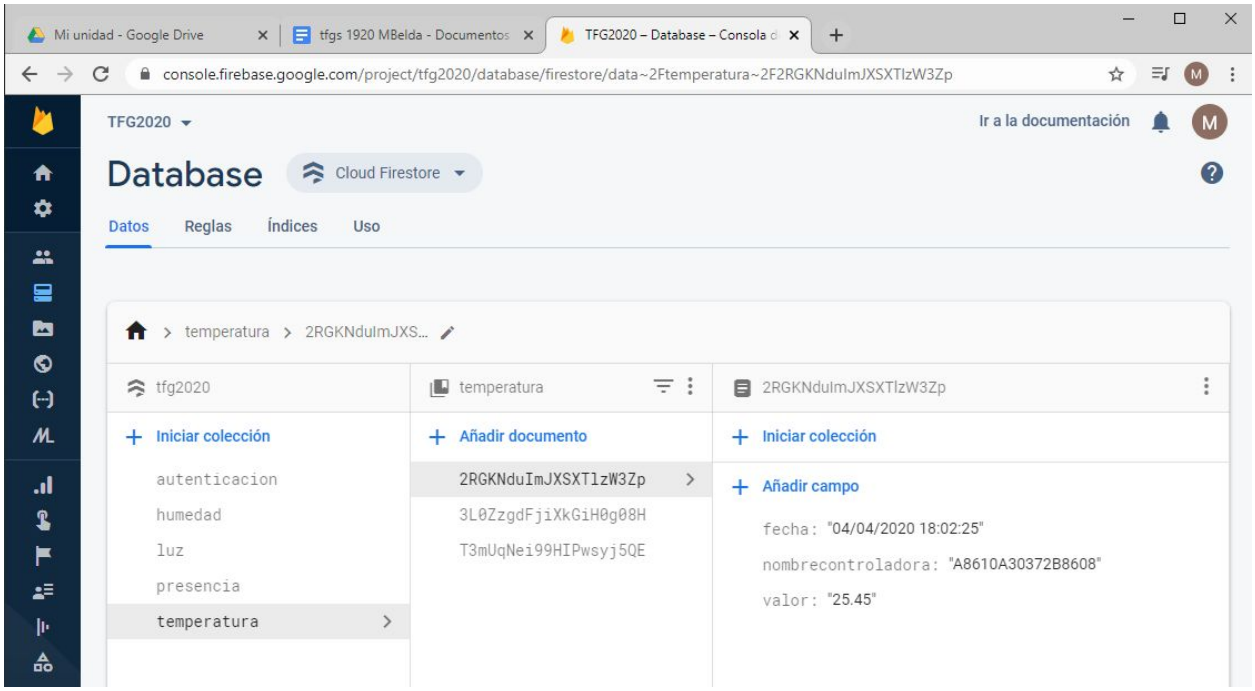


Figura 34. *Base de datos de firebase del proyecto.*

Las demás colecciones serían las respectivas a las demás categorías de sensores, con unos campos similares a los de la colección de temperatura.

Cabe aclarar que siempre que se habla de una base de datos de Firebase, se refiere a una base de datos de tipo Cloud Firestore. Esto se debe a que Firebase ofrece dos tipos de bases de datos, que son la mencionada anteriormente y Realtime Database.

Cloud Firestore ofrece más novedades que Realtime Database, ya que esta última fue la que ofrecía de manera inicial Firebase, mientras que Cloud Firestore es una base de datos más reciente y novedosa.

Podemos destacar ciertas diferencias entre ambas en la tabla 5.

Tabla 5. *Comparativa Realtime Database y Cloud Firestore [30].*

	Realtime Database	**Cloud Firestore**
Modelo de Datos	Estructura arbol JSON: ● Datos simples fáciles de almacenar ● Datos complejos difíciles de organizar.	Colecciones de documentos: ● Datos simples fáciles de almacenar ● Datos complejos fáciles de organizar, con subcolecciones dentro de documentos.
Consultas	Consultas directas con funciones de ordenamiento y filtrado limitadas.	Consultas indexadas con ordenamiento y filtrado compuestos.
Escrituras y transacciones	Operaciones básicas de escritura y transacción.	Operaciones avanzadas de escritura y transacción (transacciones desde cualquier lugar de la BD).
Escalabilidad	Necesidad de fragmentación: ● 200,000 conexiones simultáneas ● 1,000 escrituras/segundo.	Escalamiento automático: ● 1 millón conexiones simultáneas ● 10,000 escrituras/segundo.
Seguridad	Lenguaje de reglas en cascada que separa la autorización de la validación.	Reglas sin formato de cascada que combinan autorización y validación.

3.1.7 Alternativa a controladora MKR WAN 1300

Se ha realizado un segundo esquema para la red LoRa presentada en los puntos anteriores.
Se ha cambiado la controladora Arduino con conectividad LoRa por un conjunto de módulos que realizan las mismas funciones que dicha controladora.

En este caso, y dada la problemática causada por el Covid-19, se va a hablar sobre la implementación teórica de esta alternativa. No se ha podido realizar una implementación práctica debido a que no se ha podido acceder a la Universidad para soldar los componentes del módulo. Todo lo de este apartado es un supuesto teórico.

3.1.7.1 Dispositivos

En este apartado se detallan los dispositivos que conforman el nuevo front-end de la red.

- **Arduino DUE**: controladora Arduino que conformará el núcleo de los distintos módulos. A ella se conectarán todos los módulos, alimentandolos en todo momento. Además, será configurada como placa predeterminada a la hora de ejecutar los scripts de Arduino en el IDE.
- **Grove Base Shield**: controladora Grove a la cual se conectan todos los sensores. Esta controladora no tiene puerto propio de alimentación, por lo que se conecta sobre el Arduino DUE. Puede trabajar a 3.3V o 5V, dependiendo de las necesidades del entorno. Los sensores serán conectados mediante conexiones Grove a la controladora, a través de los distintos puertos analógicos y digitales de la misma.
- **Sensores**: se emplean sensores de diversos tipos. Podemos destacar los de temperatura y presencia.
- **Actuadores**: se hace uso de actuadores que trabajarán junto con los sensores. Entre los distintos actuadores destacamos LEDs, un pequeño ventilador y un zumbador.
- **Módulo LoRa**: dispositivo encargado de enviar las mediciones hacia el backend mediante la conectividad LoRa. El módulo consta de un chip RF95 sin conectividad Grove. Este ha de ser soldado a una placa de conexiones, para, posteriormente, soldar contactos a la placa.

El módulo LoRa anterior ha reemplazado al módulo LoRa Grove RF95. Este módulo servía, únicamente, para realizar una comunicación cliente-servidor junto con otro módulo del mismo modelo, pero no podía enviar datos a un backend mediante un gateway [31].

Los problemas se reducían a la compatibilidad de la librería LMIC de IBM con el módulo. Esta librería hace posible que distintos módulos LoRa puedan enviar datos a backends como The Things Network entre otros, más allá de una comunicación punto a punto.

Esta librería es solo compatible con módulos con determinadas conexiones, por lo que el módulo con conexión Grove no estaba soportado por dicha librería. Es por ello que, dicho módulo no puede emplearse para una conexión con un backend [32].

Para solucionar estos problemas, se ha seleccionado el módulo LoRa explicado en el último punto de este apartado, el cual no incorpora conexión Grove, ya que es un chip simple al cual se sueldan las conexiones que se requieran. El mapeo de pines entre el módulo y la controladora es el que se muestra en la tabla 6.

Tabla 6. *Esquema conexión módulo RF95 y Arduino DUE [33].*

Módulo RF95	Arduino DUE
GND	GND
3.3V	3.3V
DIO0	1
DIO1	2
RESET	3
NSS	4
SCK	5
MOSI	6
MISO	7

3.1.7.2 Scripts de funcionamiento

El código usado corresponde al código del apartado **8.1 Códigos - Script RF95 [32].**
El propio código contiene comentarios explicando los bloques en los que se divide.

Se ha desarrollado un script de Arduino a modo de prueba de funcionamiento de los sensores y el módulo. En el script hay un par de casos de ejemplo. Dado que hay diversos sensores y actuadores de pruebas, se han realizado aproximaciones a lo que podría ser una implementación real. Destacamos los siguientes:

- **Detector de movimiento con zumbador**: el módulo implementa un sensor de presencia que, en caso de detectar algo, encenderá un zumbador a modo de alarma.
  Cuando el sensor detecta presencia, su variable, PINMOVIMIENTO, tiene el valor a 1. La lectura de este valor se almacena en la variable salidaSensor para, posteriormente, encender y apagar el zumbador. El valor del zumbador puede ser enviado al backend para, posteriormente, ser almacenado en la base de datos de Firebase.

```
int salidaSensor = digitalRead(PINMOVIMIENTO);

if (salidaSensor == 1) {
  digitalWrite(ZUMBADOR,HIGH);
  delay(1000);
  digitalWrite(ZUMBADOR,LOW);
}
```

Figura 35. *Ejemplo 1 para el módulo LoRa RF95.*

- **Sensor de temperatura con ventilador**: en este caso, se hace uso de un sensor de temperatura que realiza mediciones cada cierto intervalo de tiempo. Si durante una medición detecta que la temperatura ha excedido un límite establecido, activa un ventilador para reducir la misma. Cuando la temperatura baja del límite, el ventilador se apaga.
  Este caso puede ser un caso real de una máquina o sistema que requiere de cierta ventilación automatizada. El código empleado es el mostrado en la figura 36.

```
unsigned long currentMillis = millis();

if (currentMillis - previousMillis >= interval) {
  previousMillis = currentMillis;
  int a = analogRead(pinTempSensor);
  float R = 1023.0 / a - 1.0;
  R = R0 * R;
  float temperature = 1.0 / (log(R / R0) / B + 1 / 298.15) - 273.15;
  SerialUSB.print("La temperatura es de: " );
  SerialUSB.println(temperature);

  if (temperature >= 23.00) {
    SerialUSB.print("Se ha encendido el ventilador por alta temperatura" );
    analogWrite(PINMOTOR, 255);
  } else
    analogWrite(PINMOTOR, 0);
}
```

Figura 36. *Ejemplo 2 para el módulo LoRa RF95.*

La condición principal se encarga de realizar las mediciones por intervalos de tiempo. La segunda condición se encarga de activar el ventilador según la medición de la temperatura.

3.1.7.3 Imágenes de los componentes

En la figura 37 se puede observar una aproximación a cómo quedaría el módulo LoRa ensamblado. El chip RF95 se ha colocado sobre la placa, y a la placa se ha soldado la antena helicoidal elegida. En cada conexión de la placa, se encuentran soldados los contactos.

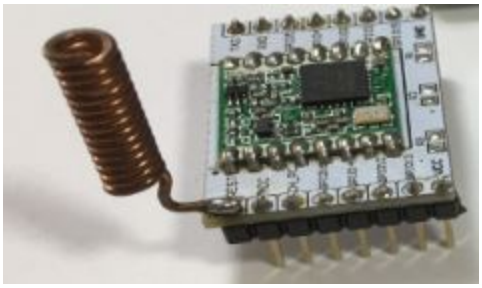


Figura 37. *Módulo LoRa RF95 ensamblado [33].*

El otro componente es el módulo de sensores, que contiene las controladoras Arduino. Este es el mostrado en la figura 38.

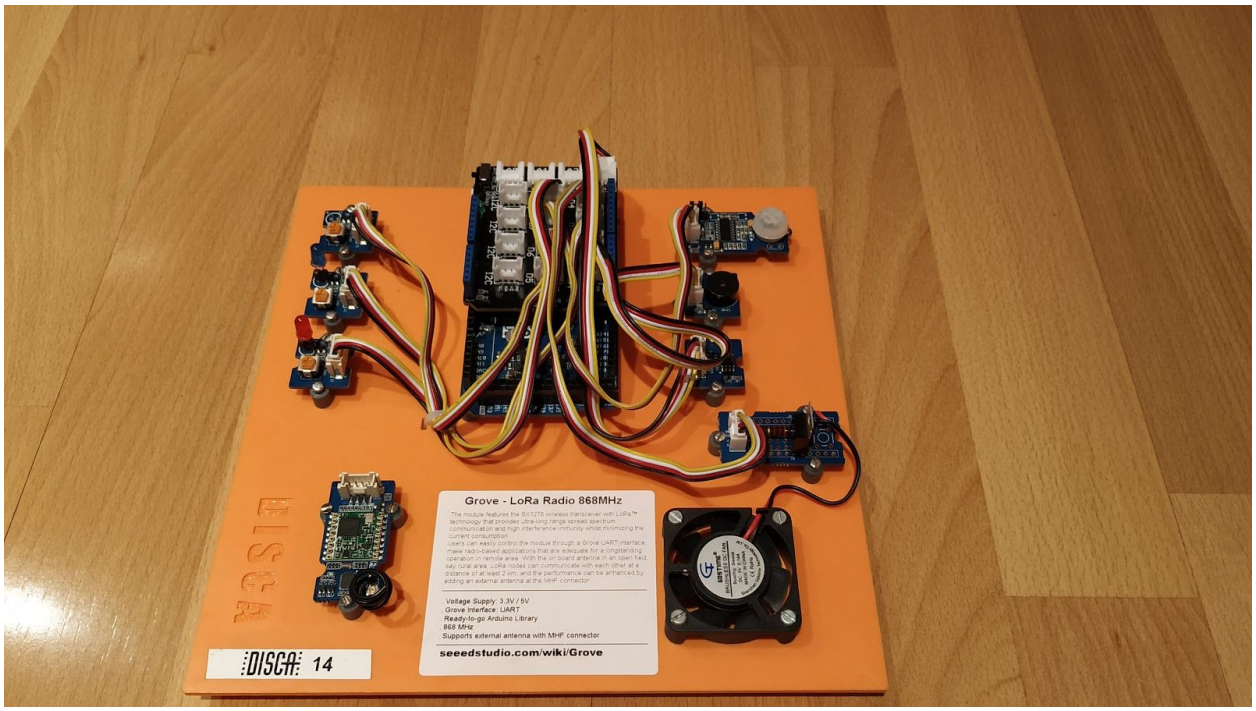


Figura 38. *Front end alternativo a controladora MKR WAN 1300.*

A este módulo se conectaría el módulo LoRa de la figura 37. Las conexiones se especifican en el apartado **3.1.7.1 Dispositivos**.

Esta sería la implementación necesaria para una sustitución de la controladora MKR WAN 1300, empleada como modelo inicial en el proyecto para la recogida y envío de datos.
Este ha sido un ejemplo de las diversas posibilidades a la hora de integrar e implementar sensores para una red IoT. Este modelo en concreto es más flexible que el modelo de la controladora MKR WAN 1300, dado que se compone de diversos módulos.

El script llamado **Script RF95** de los adjuntos del proyecto es el código al que se ha hecho referencia, y que sería usado si se hubiera podido implementar este modelo. Toda la funcionalidad del backend sería la misma, con la única diferencia de registrar los nuevos componentes sobre el backend, de la misma manera que se hizo con la controladora principal.

3.1.7.4 Registro en el backend

Al igual que el módulo MKR WAN 1300, el módulo RF95 debe ser registrado en el backend de Loriot. A diferencia del anterior módulo, este no contiene un identificador interno o DEV EUI, por lo que no puede ser registrado en base a este identificador.

En este caso, se debe registrar mediante el método de generación de parámetros, donde se le pide al propio backend que genere todos los parámetros.

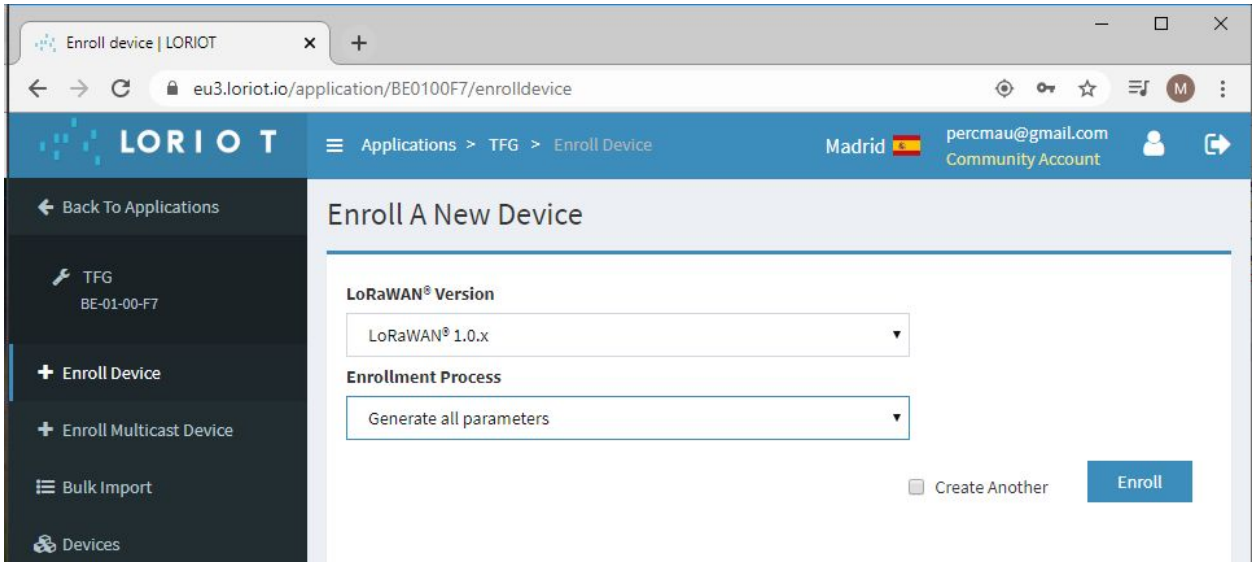


Figura 39. *Registro de un dispositivo en el panel de control del backend de Loriot.*

De esta manera, quedan registrados ambos módulos en el backend, compartiendo una misma aplicación. Al módulo que se acaba de registrar, se podría vincular y enviar datos cualquier módulo, ya que no se asocia a un identificador de algún módulo.
Es el propio backend el que establece el vínculo con el primer módulo que le envía datos, manteniendo este vínculo para futuras comunicaciones con el mismo.

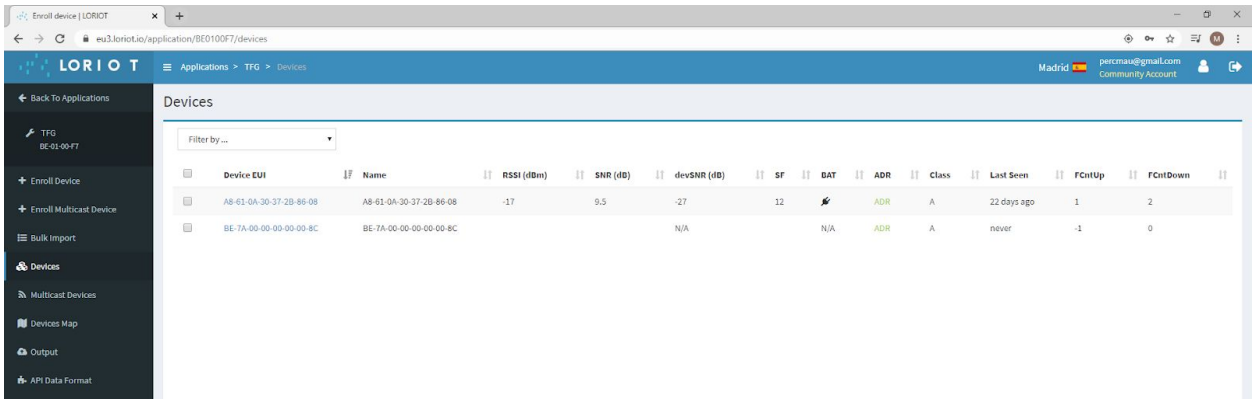


Figura 40. *Dispositivos registrados en el panel de control del backend de Loriot.*

3.2 Entorno de desarrollo de la aplicación

La aplicación ha sido desarrollada en el framework Flutter, usando el lenguaje de programación Dart. Para poder hacer uso de Flutter, se requiere su instalación en un editor de código. En este caso, el editor de código usado es Visual Studio Code.

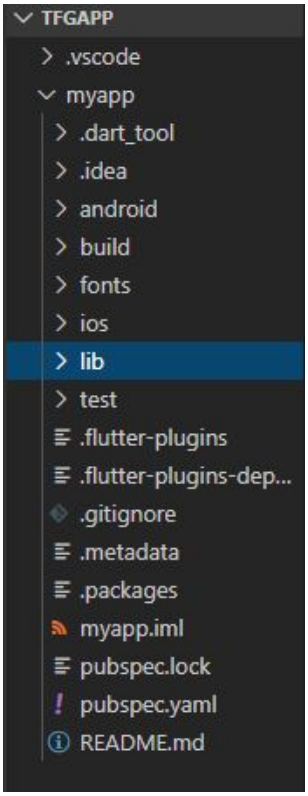
A continuación, se va a mostrar los pasos seguidos para la instalación y puesta a punto del framework sobre el editor [34]:

- Dentro del editor, ir a **View > Command Palette**
- Escribir "flutter" en el navegador y seleccionar **Instalar**
- Escribir ahora sobre el navegador "doctor", y seleccionar la opción **Flutter: Run Flutter Doctor**
- Mostrará posibles errores en la instalación de Flutter. Si los hay, hay que repararlos. Si no muestra errores, todo se ha instalado correctamente.

Una vez instalado el framework sobre el editor, se podrá crear la aplicación. Para crear la aplicación, hay que seguir unos breves pasos:

- Ir de nuevo a **View > Command Palette**
- Escribir "flutter" en el navegador y seleccionar **Flutter: New Project**
- Escribir el nombre de la aplicación y seleccionar o crear el directorio raíz del nuevo proyecto.

Con estos breves pasos, la aplicación está lista para ser usada. En la figura 41, se observa como es la estructura de la aplicación desarrollada en este proyecto.



El nombre del proyecto es "TFGAPP", y se constituye por una serie de directorios. Los directorios a destacar por su funcionalidad son [35]:

- **lib**: directorio de los ficheros .dart de nuestra aplicación, o lo que es lo mismo, las vistas de la aplicación. Cada fichero con extensión .dart contiene una vista y la funcionalidad de la vista, todo en un fichero.
- **android e ios**: son proyectos nativos de aplicaciones android e ios respectivamente. Si se va a desplegar el código sobre una plataforma android, esta se inyectará en el proyecto del directorio android. Lo mismo sucede con el sistema ios.
- **build**: contiene el resultado de la compilación de nuestra aplicación.

Figura 41. *Estructura del proyecto de la aplicación.*

Entre los ficheros del proyecto, cabe destacar **pubsec.yaml**, ya que es importante a la hora de instalar ciertas dependencias. Este fichero es el fichero de configuración del proyecto, y contiene:

- Propiedades del proyecto como el nombre, la descripción y la versión
- Dependencias del proyecto, como por ejemplo las necesarias para trabajar con bases de datos de firebase
- Imágenes u otros elementos que va a requerir la aplicación.

En este caso, el fichero de configuración contiene las dependencias a la base de datos de firebase empleada en el proyecto, y otras como dio, para aspectos de diseño.

```yaml
dependencies:
  flutter:
    sdk: flutter
  firebase_core: ^0.4.4+3
  cloud_firestore: ^0.13.4+2
  dio: ^3.0.9
```

Figura 42. *Dependencias del proyecto de la aplicación.*

En cuanto al entorno de desarrollo, no se ha requerido de ningún cambio adicional, por lo que en los siguientes apartados se detallarán aspectos de la aplicación.

El código de la aplicación corresponde al código del apartado **8.1 Códigos - Aplicación TFG.** El propio código contiene comentarios explicando los bloques en los que se divide.

3.3 Implementación práctica de la aplicación

3.3.1 Maquetas

En este apartado se exponen los diseños de las vistas realizados antes del desarrollo de la aplicación. Se ha pensado en una aplicación basada en tres vistas.

En la figura 43 se puede observar la vista principal, que se obtendrá cuando un usuario acceda a la aplicación móvil. Es una pantalla donde se restringe el acceso a los datos de la aplicación, y por consiguiente a las vistas posteriores.

Para el acceso a la aplicación, hay que ingresar la clave correcta en un campo de texto. Esta clave se encuentra almacenada en un registro de la base de datos. En caso de que el usuario introduzca la clave correcta, accederá a la aplicación. En caso negativo, permanecerá en esta misma pantalla, teniendo más intentos.

De esta manera, se mantiene la confidencialidad de los datos de nuestros sensores.

Figura 43. *Maqueta de la vista de autenticación.*

Figura 44. *Maqueta de la vista de categorías de sensores.*

Al introducir la clave correcta, se muestra la vista principal de la aplicación, es decir, la figura 44.

En esta vista, se muestran a modo de grid o cuadrícula los distintos tipos de sensores, separados según el tipo de medición que realicen.

Para acceder a una categoría de sensores, habrá que hacer tap sobre una cuadrícula en particular, llevándonos así a la siguiente vista de la aplicación.

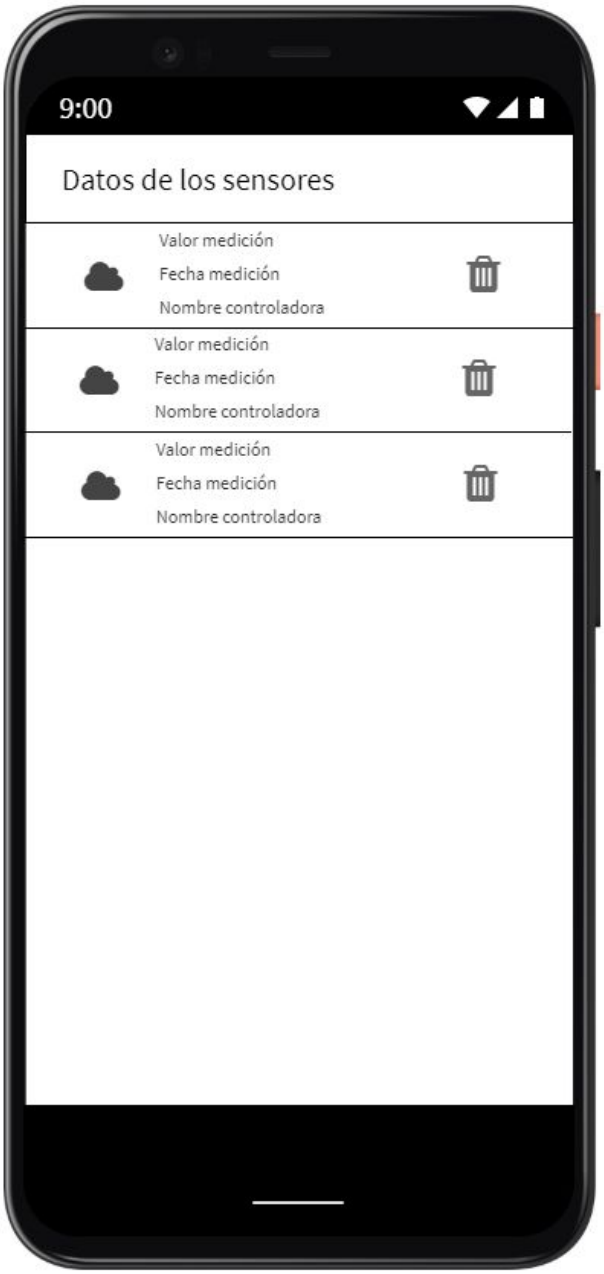


Figura 45. *Maqueta de la vista de mediciones de una categoría.*

A continuación, se muestra la última pantalla de la aplicación, la figura 45. En esta vista se muestran los valores de las mediciones de los sensores.

Al seleccionar una categoría determinada de sensores en la vista anterior, esta vista actual nos mostrará la información de todos los sensores de la red de esta categoría. De esta manera, se separa el flujo de datos por tipo de sensores, facilitando la visualización de datos al usuario.

Esta vista se constituye por una lista de elementos, donde cada elemento es una medición. Para cada medición, se muestra el valor de la misma, la fecha y hora en la que se ha realizado la medición, y la controladora Arduino que ha enviado los datos.

Junto con cada medición, se ha habilitado un botón de eliminar, que eliminará tanto de la aplicación como de la base de datos una medición, es decir, un documento de una colección específica.

El usuario podrá volver hacia la pantalla anterior mediante los botones hardware o software de su dispositivo móvil.

3.3.2 Diagramas de funcionamiento

En la figura 46, se muestra el diagrama de funcionamiento de la aplicación.

Para comprobar si la clave ingresada es correcta, se realiza una consulta a la base de datos de Firebase, aunque en el diagrama se ha obviado. Lo mismo sucede cuando se elimina un registro de la lista de mediciones de una categoría de sensores, que elimina el registro dado de la base de datos.

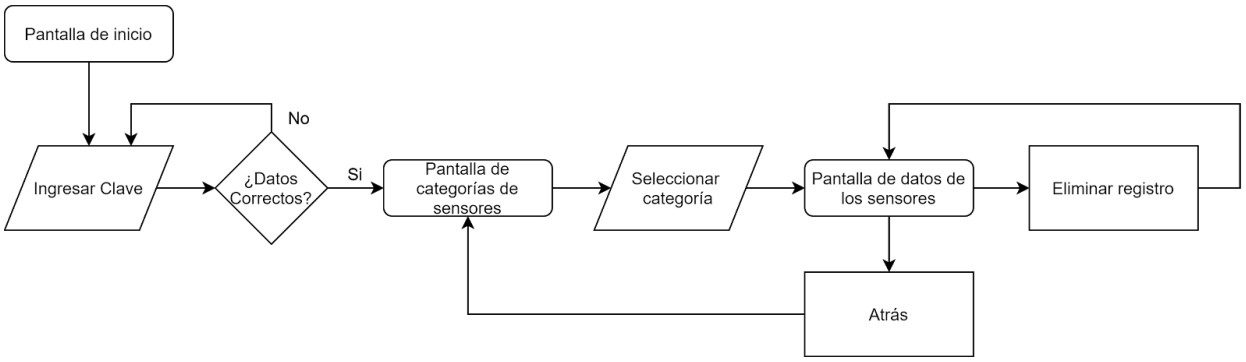


Figura 46. *Diagrama de funcionamiento de la aplicación.*

A continuación, en la figura 47, se muestra la leyenda del diagrama anterior:

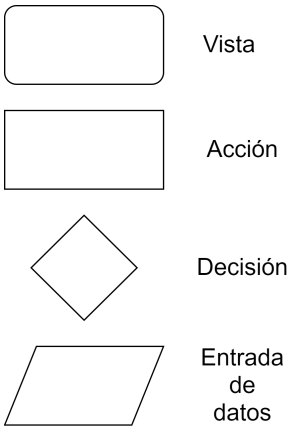


Figura 47. *Leyenda del diagrama de funcionamiento de la aplicación.*

3.4. Pruebas

En el apartado de pruebas se van a realizar tanto pruebas de funcionamiento individuales como colectivas. Estas se dividen en los siguientes apartados.

3.4.1 De sistema

Se realizarán pruebas individuales de funcionamiento a los distintos componentes que conforman la red, incluyendo la base de datos. Con estas simples pruebas se podrán detectar posibles errores, tanto a nivel electrónico como a nivel de servicio.

3.4.1.1 De la controladora Arduino, gateway y backend

Este ejemplo trata de enviar un "Hola Mundo" desde la controladora Arduino hacia el backend, y la posterior visualización del mismo desde el backend de Loriot. Con esto se probará lo siguiente:

- Funcionamiento a nivel físico de la controladora Arduino para descartar posibles problemas electrónicos
- Recepción de datos del gateway
- Recepción de datos del backend.

En cuanto a la configuración del IDE, tan solo se ha de seleccionar la placa con la que se trabaja (Arduino MKR WAN 1300) y el puerto (COM4).

En cuanto al código, se ha seleccionado un ejemplo que el plugin MKR WAN nos trae, en el cual hay que especificar dos identificadores que podemos encontrar en Loriot, así como la banda de frecuencia a la que trabajan nuestros dispositivos (los identificadores se han definido en el apartado de implementación).

Se ejecuta el código y se abre el terminal serie de Arduino. Aquí, en la figura 48, se puede observar que el sketch de Arduino nos avisa de que el mensaje ha sido enviado.

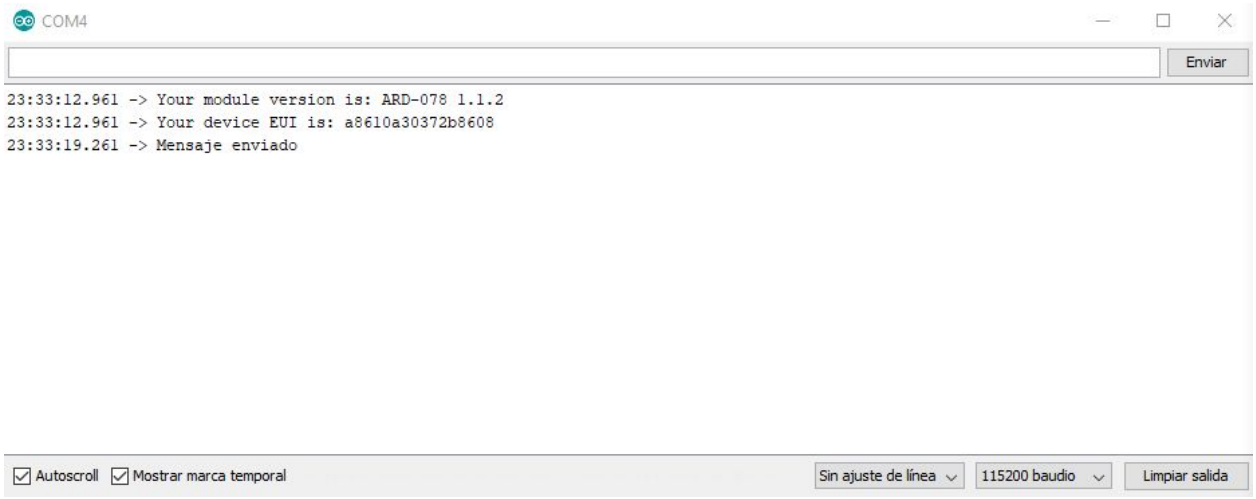


Figura 48. *Pruebas de sistema de los componentes del proyecto.*

Aunque el programa indica que el paquete se ha enviado correctamente, se procederá a observar si se ha recibido de manera correcta en el gateway, y posteriormente en el backend Loriot.

Para comprobar que el gateway ha recibido el paquete enviado, accederemos al panel de tráfico del gateway. En este panel, observamos que el gateway ha recibido el paquete, mostrándonos además el identificador o DEVEUI de la controladora que lo ha enviado. Este paquete ha pasado por las distintas fases de petición y confirmación por las que pasa un paquete al ser recibido en una red.

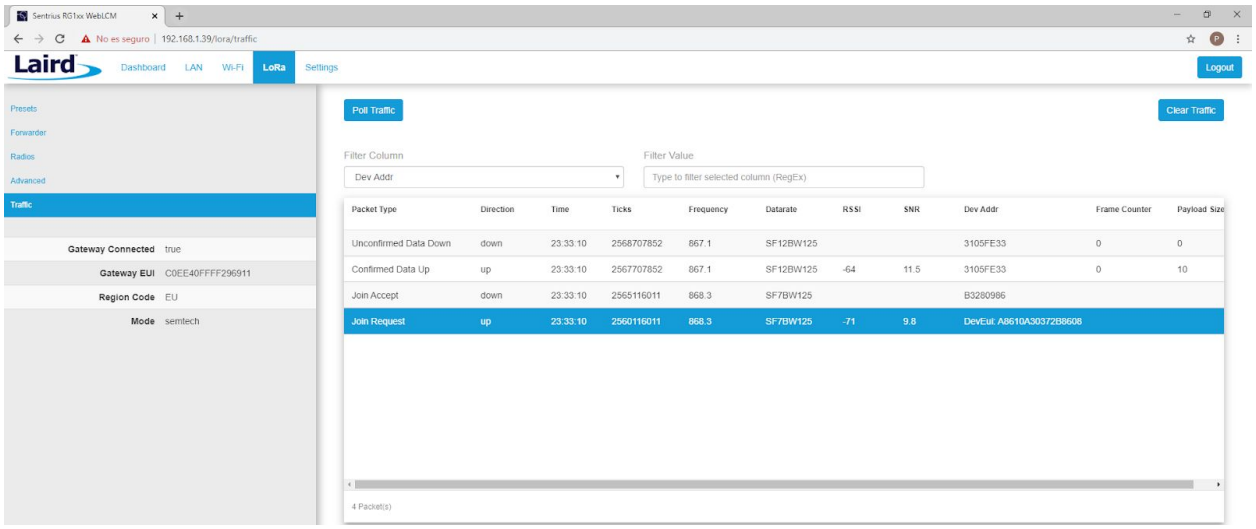


Figura 49. *Apartado LoRa del panel de control del gateway.*

Para comprobar que ha llegado al backend, se accede a nuestra aplicación dentro del mismo, y se comprueba los últimos frames o paquetes recibidos.

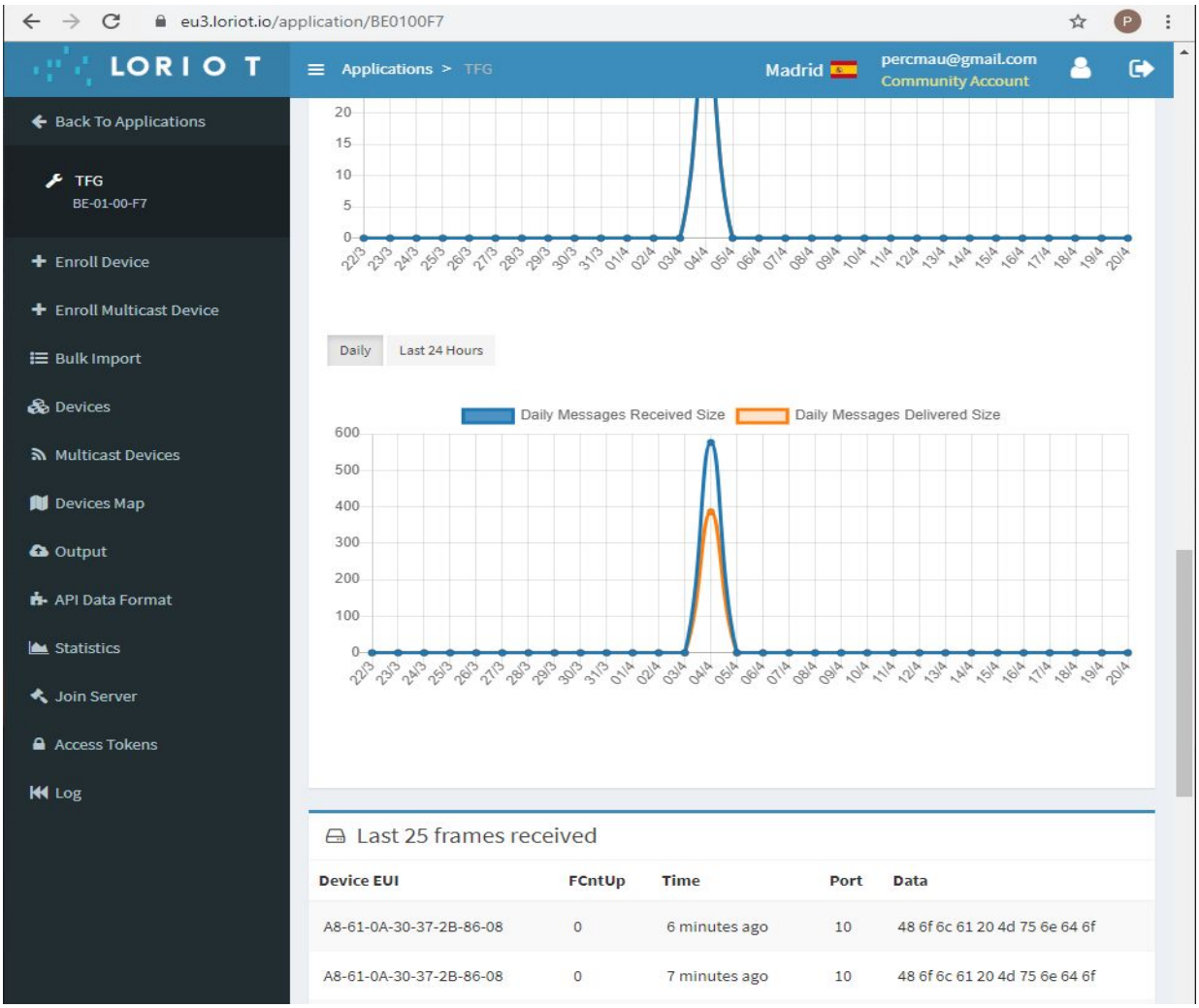


Figura 50. *Apartado de aplicación del panel de control del backend Loriot.*

Entre los distintos que hay, se puede observar el primero, que es el más reciente. Se indican distintos parámetros como el puerto por el que se ha recibido, el identificador "Device EUI" de la controladora Arduino que la ha enviado, así como el contenido del mismo, representado por el campo "Data". Se puede observar que el contenido del paquete se muestra en hexadecimal, pero si se decodifica a texto base, se puede obtener el mensaje enviado, es decir, "Hola Mundo". El código en hexadecimal es **48 6f 6c 61 20 4d 75 6e 64 6f.**

Con esta sencilla prueba, se ha confirmado el correcto funcionamiento de tres componentes de la red. Dada la dependencia de estos componentes, no se ha podido realizar una prueba individualizada a cada uno de ellos, por lo que se ha realizado una colectiva.

3.4.1.2 Del broker

Se comprobará el funcionamiento individualizado del broker de mensajería a partir de la publicación de un mensaje como “Hola Mundo”. El objetivo es publicar, desde el broker, un mensaje en un tópico, y observar como, desde el propio broker, este se recibe de manera correcta.

Para este ejemplo, se hace uso de la herramienta Websocket incluida en el backend, y explicada con anterioridad.

Como se observa en la figura 51, se ha publicado el mensaje “Hola Mundo” sobre el tópico /pruebas. Desde el panel de mensajes recibidos, observamos que se ha recibido el mensaje correctamente. Esto se debe a que el broker se ha suscrito correctamente al tópico al cual hemos publicado el mensaje de prueba.

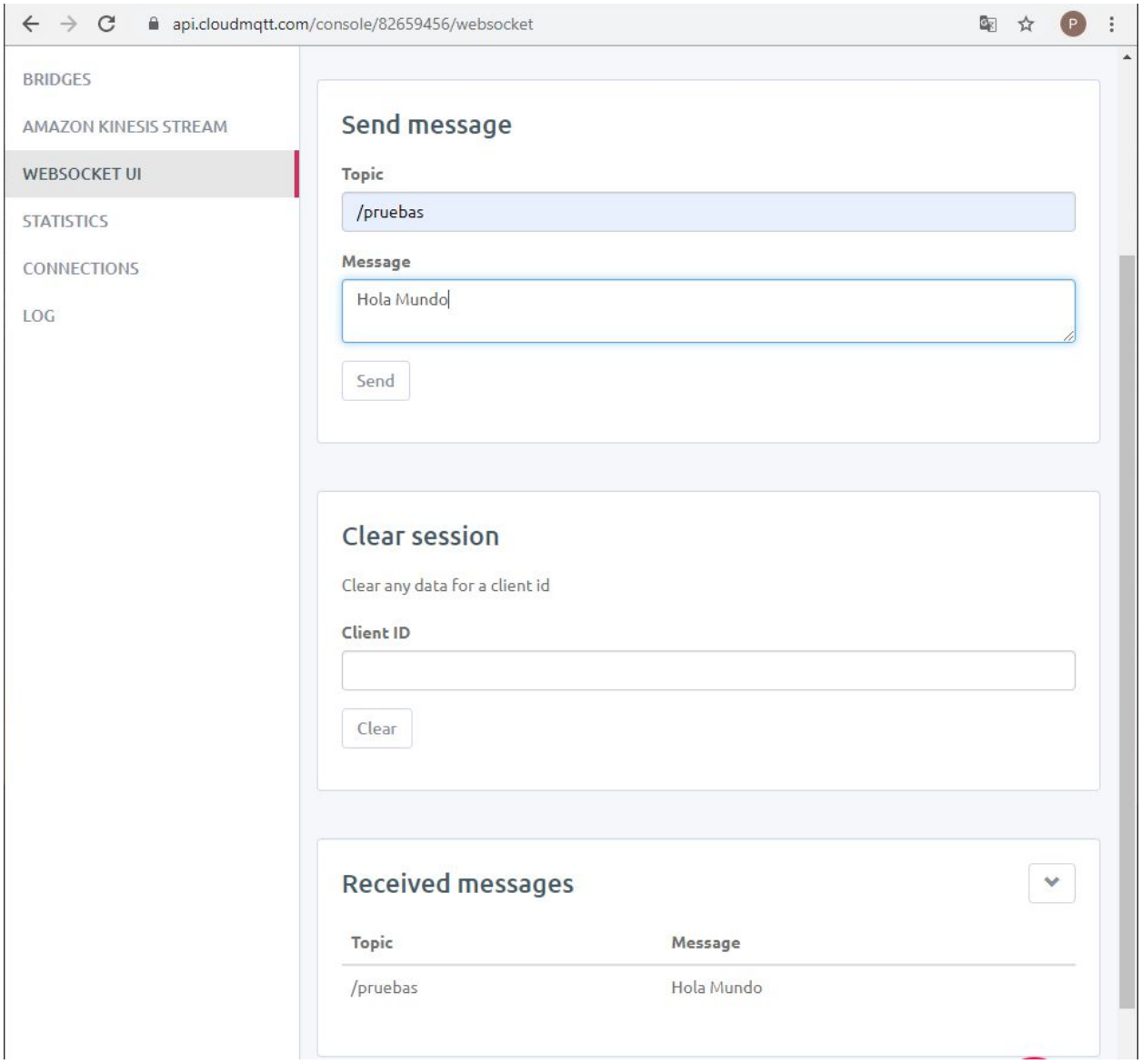


Figura 51. *Pruebas de sistema del broker del proyecto.*

Esta prueba ha revelado que el funcionamiento básico del broker es correcto, y está listo para realizar su tarea en el proyecto.

3.4.1.3 De la base de datos

Se van a realizar una serie de inserciones, modificaciones y eliminaciones de la base de datos para probar su funcionamiento. Estos cambios se realizarán desde el propio cliente que ofrece Firebase.

El estado inicial de la base de datos es el mostrado en la figura 52.

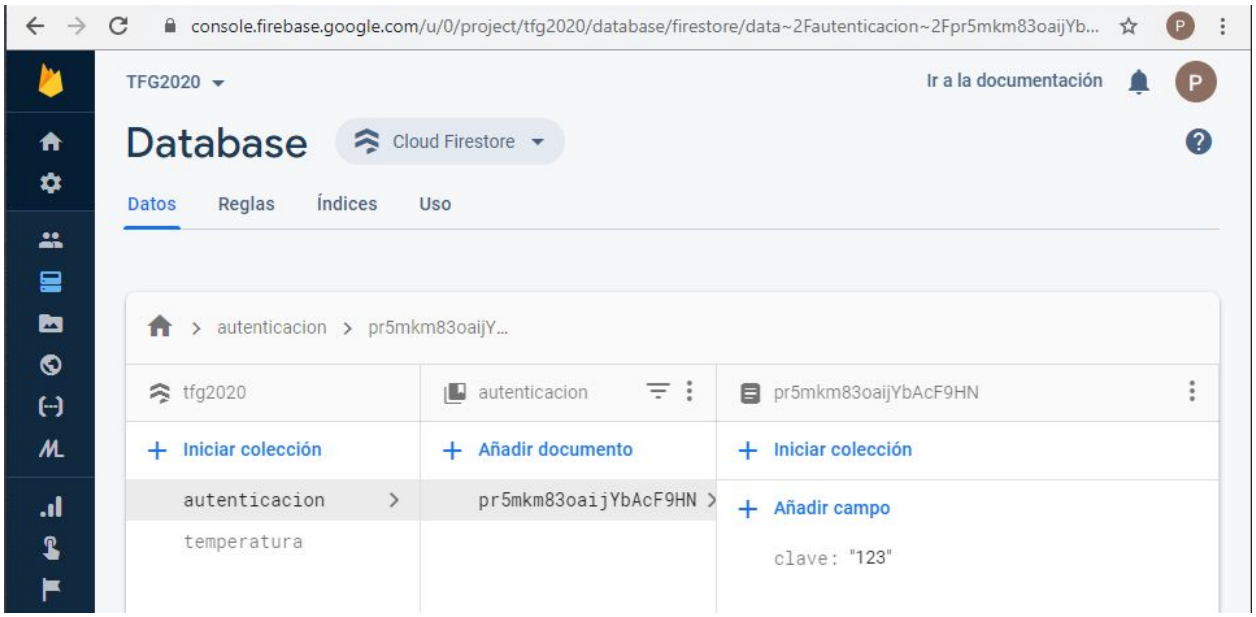


Figura 52. *Pruebas de sistema de la base de datos del proyecto.*

Como primera prueba, se creará una colección, con un documento en su interior que contendrá un campo. La colección se llama “prueba”.

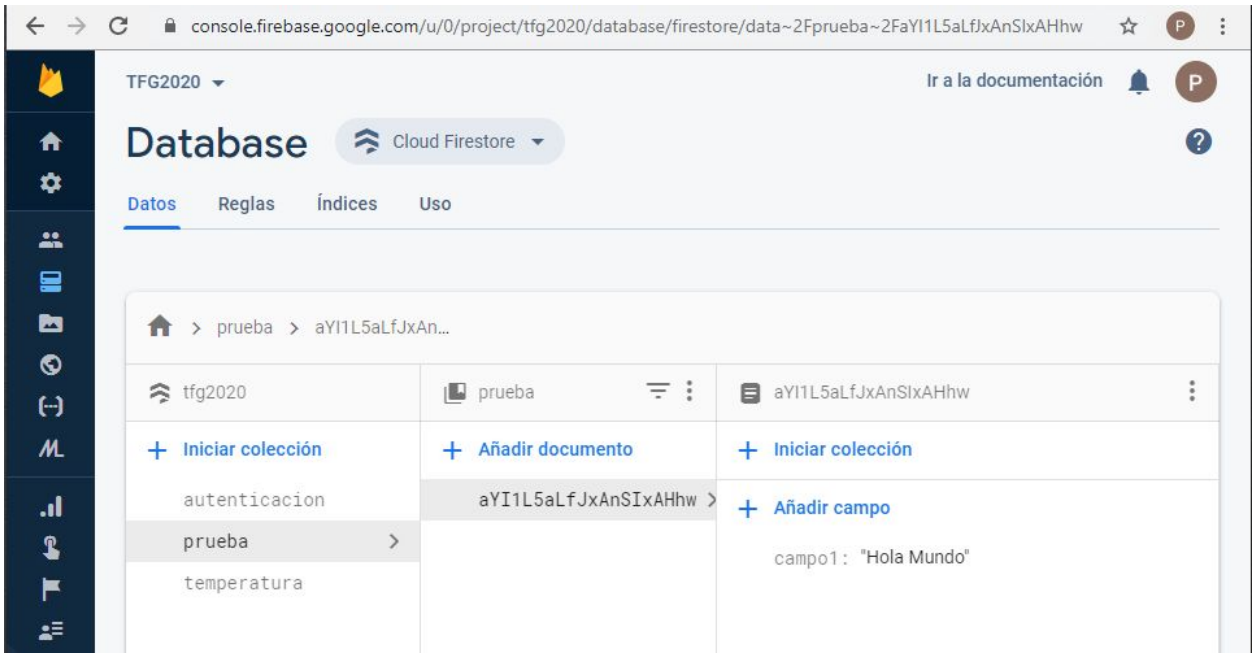


Figura 53. *Pruebas de sistema de la base de datos del proyecto.*

A continuación, se modificará el valor del campo “campo1”. El nuevo valor será “Hola Mundo 2”.

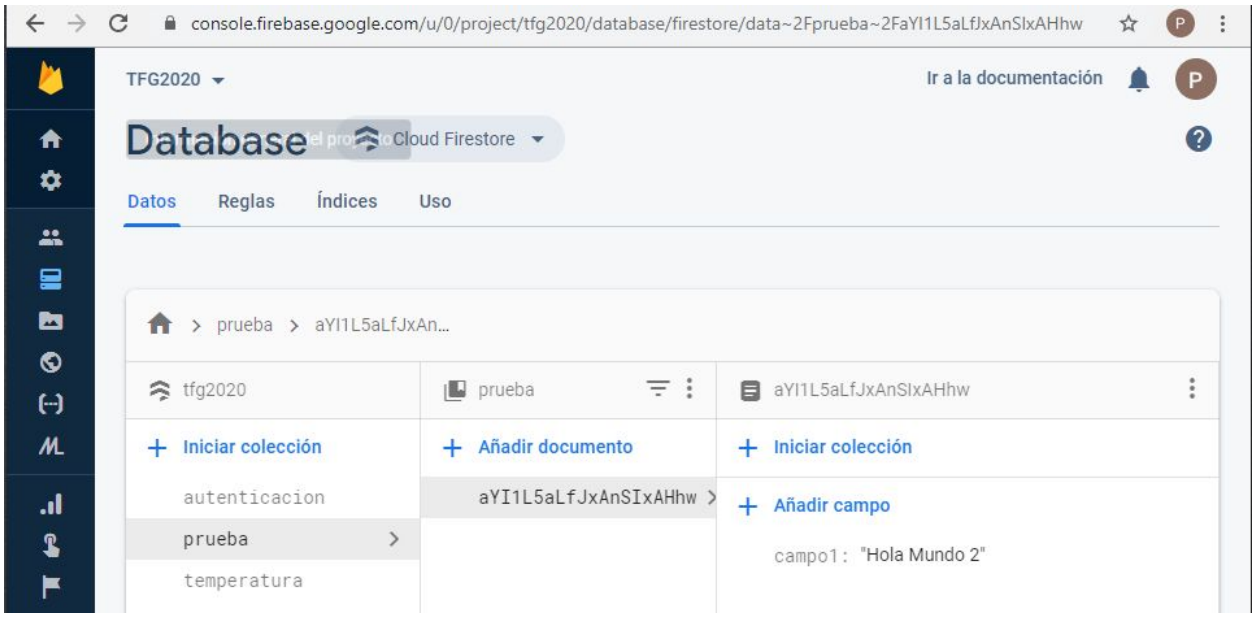


Figura 54. *Pruebas de sistema de la base de datos del proyecto.*

Por último, se eliminará la colección, con todo lo creado anteriormente en su interior. El resultado se muestra en la figura 55.

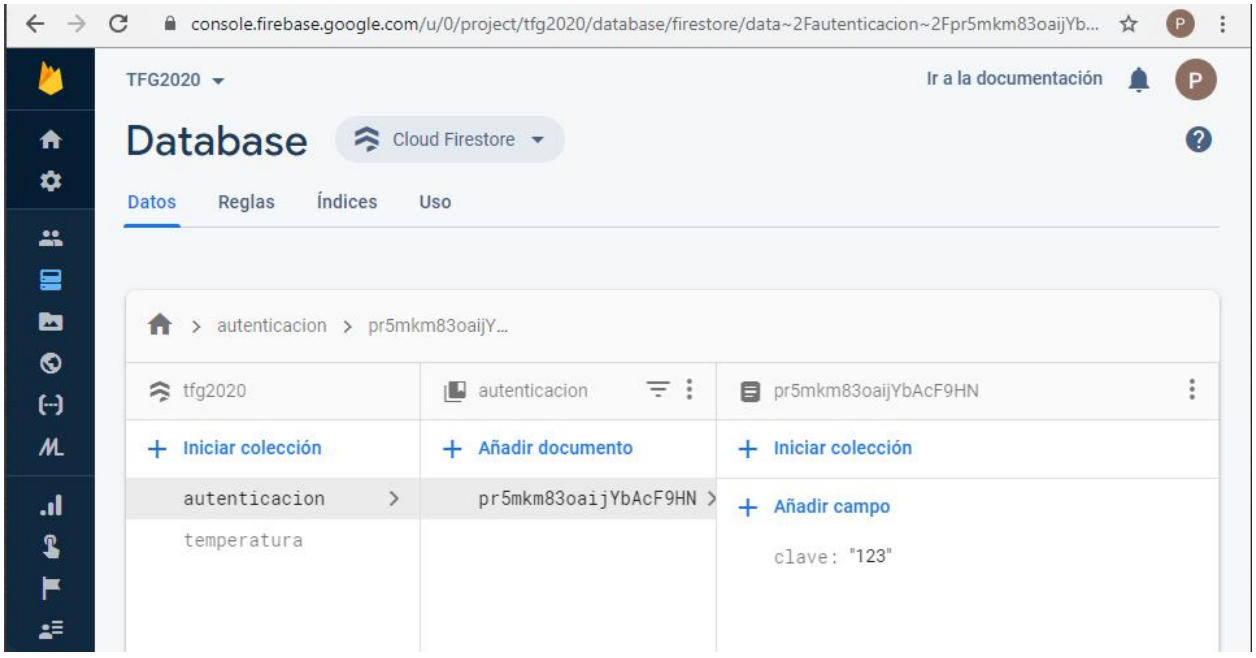


Figura 55. *Pruebas de sistema de la base de datos del proyecto.*

Con estas sencillas pruebas, se valida el correcto funcionamiento de la base de datos. Esta se encuentra lista para cumplir su función en el proyecto.

Como todas las partes principales del proyecto funcionan de manera correcta, se pasará al siguiente punto, la prueba de integración del sistema.

3.4.2 De integración de sistemas

En las pruebas de integración se comprobará la correcta integración de todos los componentes de nuestro proyecto. En este caso, se probará a enviar un dato de un sensor de temperatura desde la controladora Arduino para que, finalmente, se almacene en la base de datos.

En este caso, se mostrarán capturas de el recorrido del paquete de datos hasta llegar al destino final, pasando por todos los componentes de la red.

El esquema físico es el de la figura 56, donde se incluye la controladora Arduino con las conexiones al sensor de temperatura.

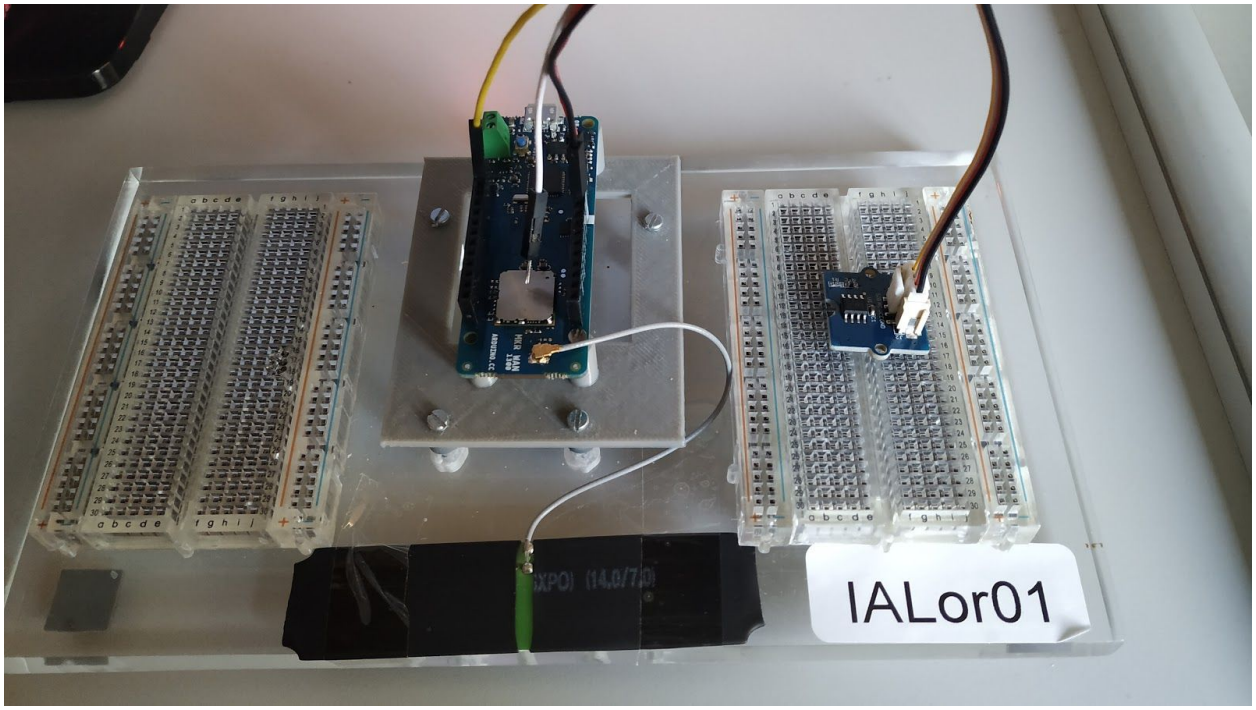


Figura 56. *Esquema físico de la controladora MKR WAN 1300 del proyecto.*

En primer lugar se ha conectado el sensor de temperatura a la controladora LoRa a través de un conector Grove. Este tipo de conector está compuesto por cuatro cables distintos. Ordenados por colores, son los siguientes:

1. Negro: conexión a tierra. Se conecta al pin GND de la controladora.

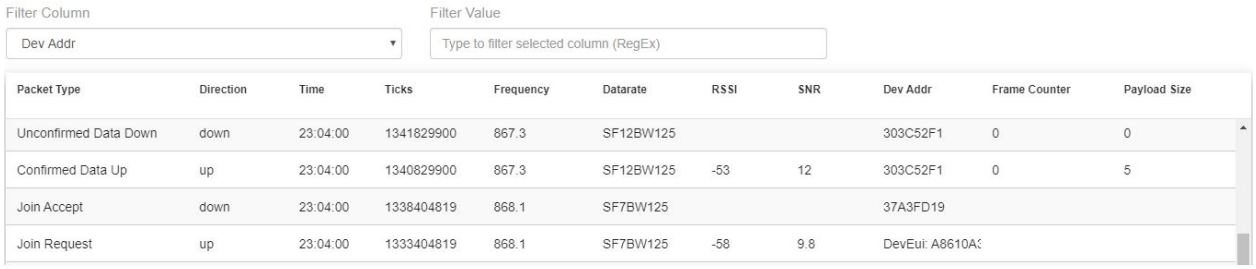
2. Rojo: VCC. Se conecta al pin VCC o 5V de la controladora. En este caso se ha conectado al pin de 5V como especifica el fabricante.
3. Blanco: NC. Para este sensor no es necesario conectar dicho cable.
4. Amarillo: SIG. Se conecta a un pin analógico de la controladora para la transferencia de datos. Se ha conectado al pin analógico A0.

En segundo lugar, se ha tomado un código de ejemplo del fabricante para probar el correcto funcionamiento del sensor.
El código usado corresponde al código del apartado **8.1 Códigos - Script MKR WAN 1300 [24].** Este código básico del sensor ha sido integrado en el código básico de envío de datos al backend usado en otros apartados anteriores.
El propio código contiene comentarios explicando los bloques en los que se divide.

Con la figura 57, se puede comprobar la correcta medición y envío de los datos del sensor de temperatura hacia el backend. En este caso, la temperatura es de 21.98 ºC.

Figura 57. *Pruebas de integración de todos los componentes del proyecto.*

Si observamos el tráfico que ha obtenido el gateway durante el envío de la temperatura de la controladora, observaremos que el gateway ha recibido el paquete de datos, y lo ha enviado al backend. Ha sido enviado por la controladora cuyo **DevEui** es **A8610A3072B8608,** que coincide con el identificador de nuestra controladora. La hora de recepción del paquete coincide con la hora de envío del paquete desde la controladora.



Packet Type	Direction	Time	Ticks	Frequency	Datarate	RSSI	SNR	Dev Addr	Frame Counter	Payload Size
Unconfirmed Data Down	down	23:04:00	1341829900	867.3	SF12BW125			303C52F1	0	0
Confirmed Data Up	up	23:04:00	1340829900	867.3	SF12BW125	-53	12	303C52F1	0	5
Join Accept	down	23:04:00	1338404819	868.1	SF7BW125			37A3FD19		
Join Request	up	23:04:00	1333404819	868.1	SF7BW125	-58	9.8	DevEui: A8610A3		

Figura 58. *Pruebas de integración del gateway del proyecto.*

Al acceder a la aplicación del backend, en los últimos paquetes de datos recibidos, observamos el paquete con la medición.

El valor de la carga del último paquete recibido por Loriot es **32 31 2e 39 38,** que pasado a texto base significa 21.98. Esta es la última medición enviada por el sensor de temperatura, por lo que el envío y recepción de datos ha sido correcta. El identificador de la controladora sigue siendo el correcto.

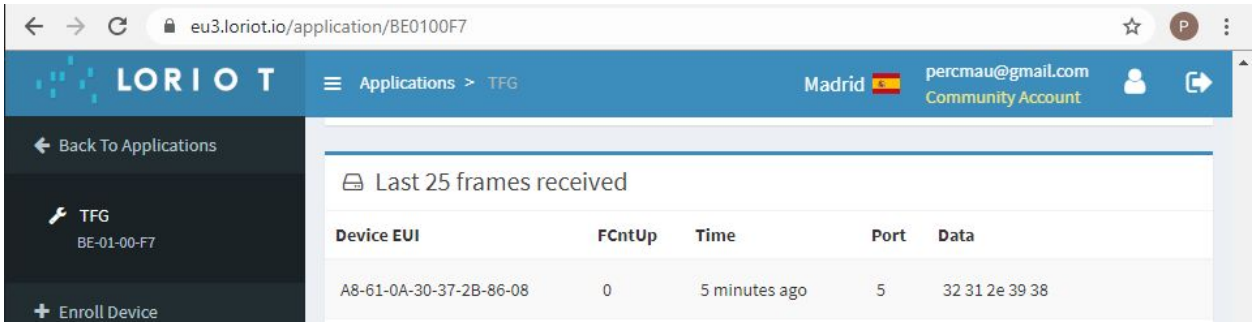


Figura 59. *Pruebas de integración del backend del proyecto.*

El siguiente componente por el que transcurre el paquete de datos es el broker, establecido como salida del backend. Desde el broker, también observaremos el estado del paquete, aunque en este caso en formato JSON.

En el paquete recibido ("cmd":"rx"), se observan los datos del paquete. Entre ellos, destacamos el valor del campo "data", que coincide con el valor de la medición inicial en formato hexadecimal. El campo EUI es el identificador de la controladora.

Figura 60. *Pruebas de integración del broker del proyecto.*

A continuación, los datos en formato JSON son tomados por el script de python. En la figura 61, se observa la consola del script, que nos indica que ha recibido correctamente los datos de una medición de temperatura, así como la correcta introducción de los datos en la base de datos.

La inserción en la base de datos se realiza después del tratado de los datos.

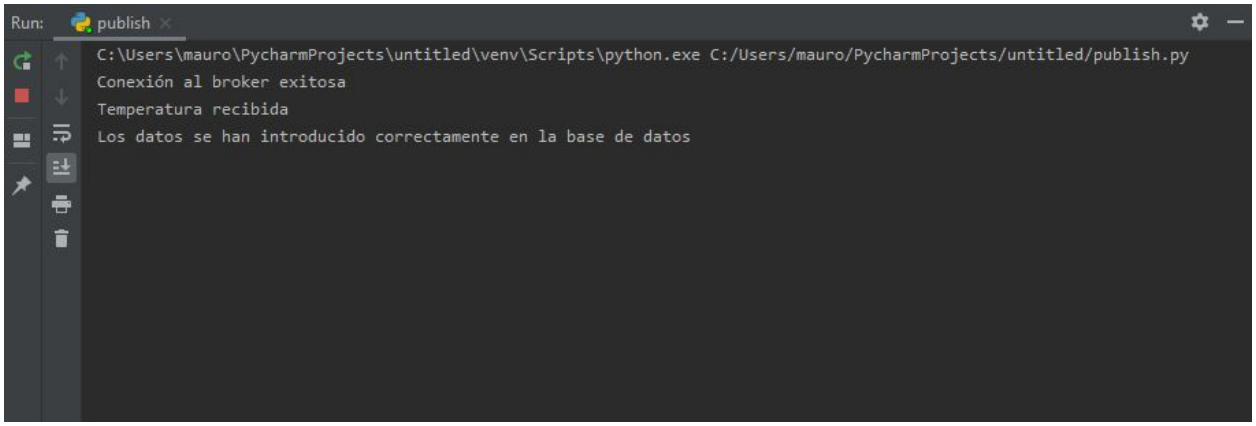


Figura 61. *Pruebas de integración del script Python del proyecto.*

Por último, el paquete es insertado en la base de datos de Firebase, en una colección llamada "pruebaintegracion", con un documento que alberga distintos campos con datos de la medición.

Podemos observar la fecha y hora de la llegada de los datos al script de python (tan solo dos segundos más del envío), así como el nombre de la controladora que lo ha enviado y el valor de la medición. El nombre de la controladora se ha establecido a partir del script de python para

una aproximación a un entorno real. El valor de la medición es 21.98, que refiere a la medición inicial tomada por nuestro sensor de temperatura.

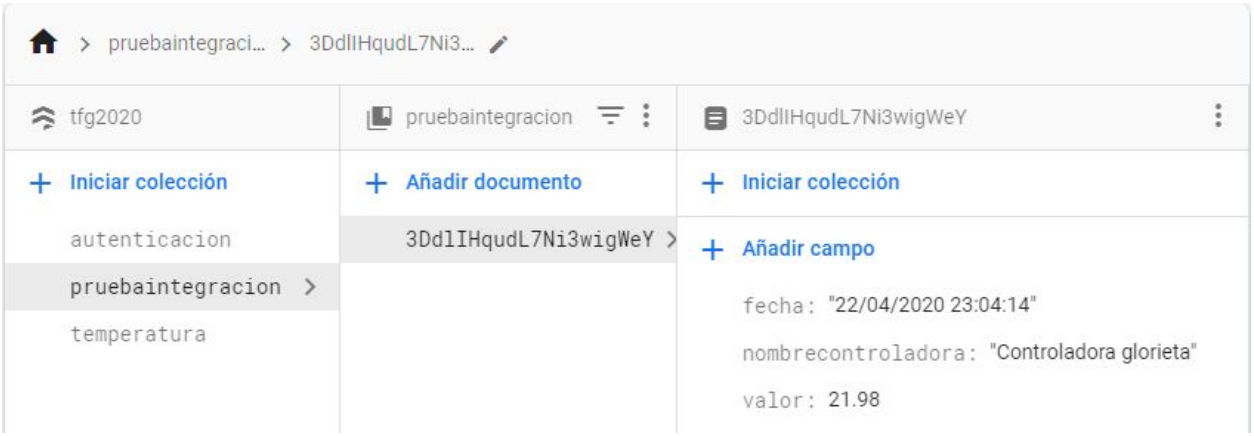


Figura 62. *Pruebas de integración de la base de datos del proyecto.*

Con estas pruebas, se ha validado la correcta integración de los componentes del sistema.
Se han obviado realizar pruebas de visualización de los datos en la aplicación ya que se probarán en apartados posteriores, con la explotación del sistema.

3.4.3 De volumen

Las pruebas de volumen tratan de sobrecargar el sistema para tratar de comparar el rendimiento teórico con el rendimiento práctico. Con unas simples pruebas se observan los máximos del sistema.

En este caso, se ha intentado realizar un envío masivo de mediciones desde la controladora Arduino hacia el backend Loriot. De esta manera, se podría observar el rendimiento del backend y de los demás componentes de la red.
No obstante, al existir una limitación en el software de la controladora, estan pruebas no se han podido realizar. El software limita el envío de mensajes a tan solo 2 por minuto, por lo que no se pueden enviar grandes cargas a los componentes.

Como alternativa, se ha empleado un software externo. Este software es Postman [36], que nos permite enviar peticiones http a componentes para evaluar su funcionamiento.
No obstante, este software presenta un problema. La licencia gratuita, no permite el envío masivo de mensajes a nuestro backend, ni la automatización del envío de los mismos.

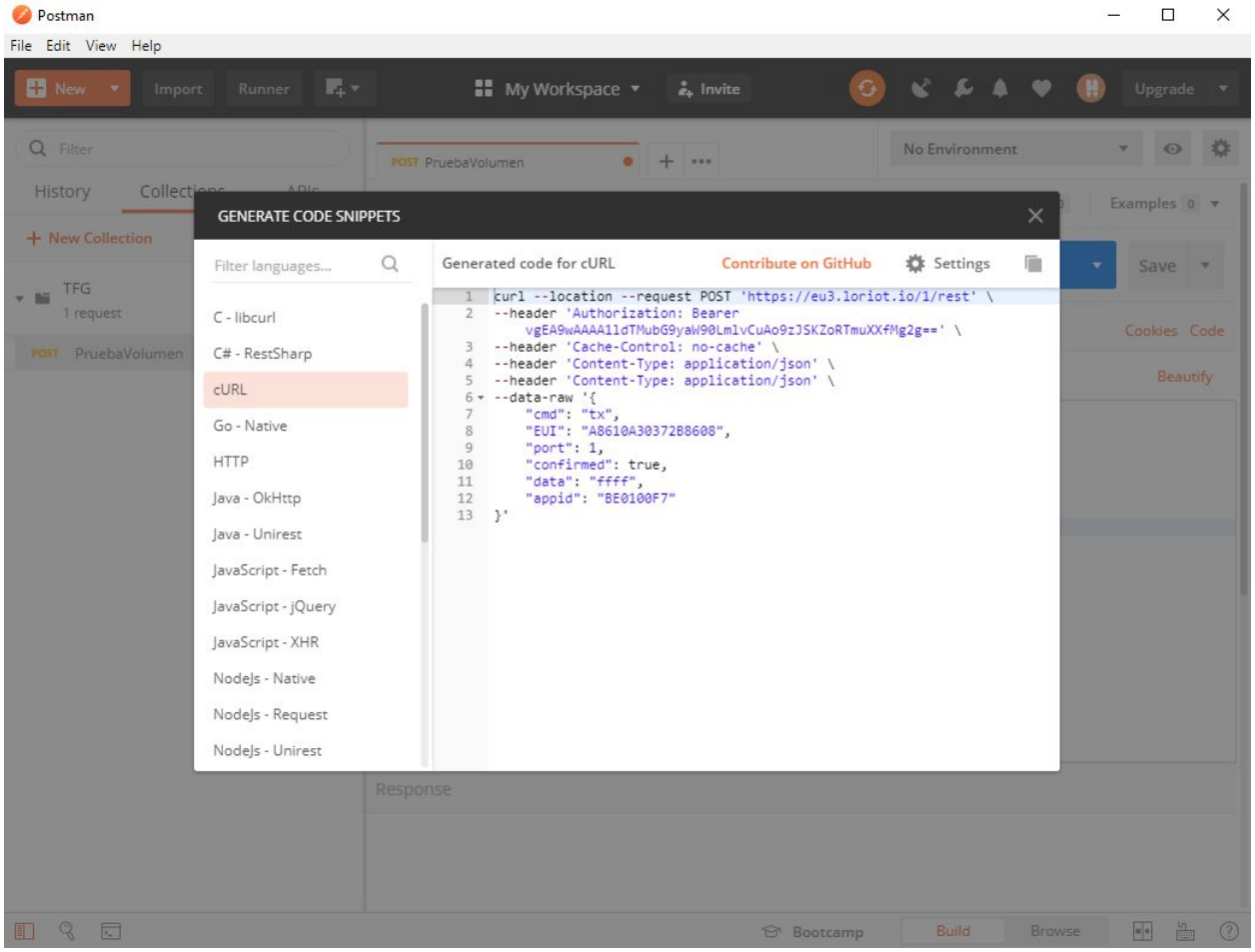


Figura 63. *Pruebas de volumen del sistema del proyecto mediante Postman.*

A modo de prueba y de forma manual, se han podido enviar en un intervalo de 1 minuto, 130 peticiones POST con mensajes simples hacia el backend. En este caso, el backend las ha recibido sin problema, y sin retrasos en la recepción de los mismos. Es por esto que, se ha estimado, que el backend puede hacer frente a mayores cargas, no obteniendo con estas pruebas su límite máximo.

Local Time		Event	Message
2020-04-28	21:16:18.977	DATA_DOWN_QUEUED	Application downlink queued encrypted: false, confirmed: true | "appid":BE0100F7,"deveui":"A8610A30372B8608","seqno":160,"instance":"nwk-1"
2020-04-28	21:16:18.74	DATA_DOWN_QUEUED	Application downlink queued encrypted: false, confirmed: true | "appid":BE0100F7,"deveui":"A8610A30372B8608","seqno":159,"instance":"nwk-1"
2020-04-28	21:16:17.780	DATA_DOWN_QUEUED	Application downlink queued encrypted: false, confirmed: true | "appid":BE0100F7,"deveui":"A8610A30372B8608","seqno":158,"instance":"nwk-1"
2020-04-28	21:16:17.422	DATA_DOWN_QUEUED	Application downlink queued encrypted: false, confirmed: true | "appid":BE0100F7,"deveui":"A8610A30372B8608","seqno":157,"instance":"nwk-1"
2020-04-28	21:16:17.108	DATA_DOWN_QUEUED	Application downlink queued encrypted: false, confirmed: true | "appid":BE0100F7,"deveui":"A8610A30372B8608","seqno":156,"instance":"nwk-1"
2020-04-28	21:16:16.762	DATA_DOWN_QUEUED	Application downlink queued encrypted: false, confirmed: true | "appid":BE0100F7,"deveui":"A8610A30372B8608","seqno":155,"instance":"nwk-1"
2020-04-28	21:15:27.439	DATA_DOWN_QUEUED	Application downlink queued encrypted: false, confirmed: true | "appid":BE0100F7,"deveui":"A8610A30372B8608","seqno":36,"instance":"nwk-1"
2020-04-28	21:15:26.856	DATA_DOWN_QUEUED	Application downlink queued encrypted: false, confirmed: true | "appid":BE0100F7,"deveui":"A8610A30372B8608","seqno":35,"instance":"nwk-1"
2020-04-28	21:15:26.206	DATA_DOWN_QUEUED	Application downlink queued encrypted: false, confirmed: true | "appid":BE0100F7,"deveui":"A8610A30372B8608","seqno":34,"instance":"nwk-1"
2020-04-28	21:15:25.812	DATA_DOWN_QUEUED	Application downlink queued encrypted: false, confirmed: true | "appid":BE0100F7,"deveui":"A8610A30372B8608","seqno":33,"instance":"nwk-1"
2020-04-28	21:15:21.52	DATA_DOWN_QUEUED	Application downlink queued encrypted: false, confirmed: true | "appid":BE0100F7,"deveui":"A8610A30372B8608","seqno":32,"instance":"nwk-1"
2020-04-28	21:15:20.810	DATA_DOWN_QUEUED	Application downlink queued encrypted: false, confirmed: true | "appid":BE0100F7,"deveui":"A8610A30372B8608","seqno":31,"instance":"nwk-1"

Figura 64. *Logs de la herramienta Postman en las pruebas de volumen.*

Aunque las pruebas anteriores no hayan sido satisfactorias, se pueden examinar los recursos que aporta cada licencia gratuita de cada componentes de la red. Con esta aproximación se pueden tratar de localizar posibles máximos de carga que puede soportar algún componente, realizando así una estimación general para el sistema. Se examinarán los recursos de:

- **Backend Loriot:** en las especificaciones del backend se puede observar que no existe un número máximo de mensajes que este pueda recibir. Esto se aplica a las tres licencias que se ofrecen, incluida la gratuita.

Figura 65. *Especificaciones de la licencia gratuita del backend.*

Se puede deducir que, a niveles teóricos, el backend no impone restricciones de carga, aunque a niveles prácticos no se ha podido demostrar.

- **Broker CloudMQTT:** dentro de las licencias que ofrece el broker, se ha examinado la licencia gratuita, que se ha empleado en este proyecto.
Dentro de la licencia, hay un parámetro que puede afectar a la cantidad de mensajes que se pueden gestionar. Esta es la cantidad de Kbits que el broker puede gestionar cada segundo. Esta cantidad son 10 Kbit/s.

Figura 66. *Especificaciones de la licencia gratuita del broker.*

Siguiendo las estadísticas que ofrece el broker, se ha estimado que un paquete de una medición puede tener un tamaño de 586 bytes. Esto es el equivalente a 4688 bits.
En este supuesto, el broker no aceptaría más de dos mediciones en un mismo segundo, ya que solo acepta hasta 10,000 bits. En el caso de que diversas controladoras envíen más de dos mediciones en un segundo, el broker podría rechazar procesar alguna de ellas, o posponerla en el tiempo.

El cálculo se ha extraído a partir de la división de la cantidad de bytes y de la cantidad de paquetes recibidos por el broker.

- **Base de datos Firebase:** la base de datos es el último elemento a observar. Como datos relevantes, se puede tomar la capacidad máxima de datos y el número máximo de escrituras diarias.

Figura 67. *Especificaciones de la licencia gratuita de la base de datos.*

En cuanto a la capacidad máxima, esta es de 1 GiB en total. Si se albergara un número muy alto de mediciones en ella (varios miles de mediciones), se podría dar un problema de capacidad, necesitando de una licencia de pago.

En cuanto al número máximo de escrituras diarias, se podrían escribir 20,000 mediciones al día en la base de datos. Este valor es muy elevado para una red con tan solo 10 dispositivos, por lo que tampoco es un factor clave en el desarrollo de la red.

En estas comprobaciones anteriores se han podido analizar ciertos límites teóricos de la red. El componente que más limitaciones puede aportar es el broker. Esto se debe a que puede gestionar un número limitado de paquetes en un mismo instante de tiempo. Los demás componentes, limitan en menor medida la red.
Con esto se puede decir que la red no debería de escasear prácticamente en recursos, y que su funcionamiento debería de ser el correcto, cumpliendo su cometido inicial.

4 Resultados

4.1 Migración al entorno de producción

Una vez comprobado el correcto funcionamiento del sistema, se requerirá realizar una migración de algunos componentes. Se dividirá en dos apartados, uno de ellos para la

migración de unos pocos componentes de la red de 10 dispositivos, y el otro para explicar los componentes que definirán la red de 10,000 dispositivos, enfocada a producción.

Red no enfocada a producción (10 dispositivos): el desarrollo de la red realizado en el apartado **3. Implementación**, puede ser considerado como un entorno de producción, y que prácticamente todos los componentes son albergados en hostings.
A esto se exceptúa el script de Python que capta datos del broker y los almacena en la base de datos, que funciona de manera local. En este apartado, se pretende migrar dicho script a una máquina virtual, para una ejecución del mismo las 24 horas del día, ya que el script debe escuchar al tópico de datos todo el tiempo.

Se pretende hacer uso de las máquinas virtuales que ofrece Google Cloud Platform, donde, de manera gratuita, durante 1 año, se puede hacer uso de diversas máquinas virtuales. En este caso, se crearía una máquina virtual con una distribución Linux, como por ejemplo Ubuntu. Esta máquina no necesitaría de muchos recursos, ya que con 2 GB de ram y 1 vCPU es suficiente para ejecutar el script.

Se ha tratado de realizar la migración a la máquina virtual, pero se ha encontrado un problema a la hora de instalar las librerías de las que depende el script. Una de ellas es la de firebase.
No se han podido probar otras distribuciones como Windows, ya que estas son de pago en la plataforma. No obstante, si se resuelve el problema de las dependencias, el código debería funcionar correctamente y de forma automática.

Con esta pequeña migración, el sistema quedaría, en su mayor parte, automatizado, obviando la pequeña implementación Arduino, que debe ser mantenida físicamente.
La aplicación se instala de forma manual sobre los dispositivos sobre los que tiene que ejecutarse, por lo que no se requiere una migración de la misma a un entorno de producción.

Red enfocada a producción (10,000 dispositivos): este apartado se centra en definir qué licencias y componentes se obtendrán para el correcto funcionamiento de una red en producción. Se usarán, si se puede, las mismas plataformas que se han empleado en la red de 10 dispositivos. Esta red se divide en los siguientes apartados:

- **Backend Loriot**: con una red tan grande, se necesitará hacer uso de licencias no gratuitas. En el caso del backend, se puede elegir entre dos de ellas, una profesional y una privada. No obstante, la licencia profesional, solo ofrece soporte para 2.500 dispositivos, por lo que no es una licencia útil en este caso.
  Es por ello que, la licencia necesaria, es la privada. En esta licencia, Loriot ofrece todas las características disponibles, ya que se enfoca al desarrollo de grandes sistemas de producción. Destacamos [3]:
  - Registro ilimitado de gateways
  - Despliegue de nube a nivel internacional
  - Registro ilimitado de dispositivos

- Número de mensajes ilimitados.

Para el despliegue del mismo, habría que contactar con la empresa y pedir presupuesto, ya que esta se encarga del diseño de la red, acorde a las necesidades.

- **Plataforma Amazon AWS**: para una mayor centralización y facilidad de uso de los componentes de la red, se emplearían los servicios de la plataforma de Amazon. Una parte de estos se enfocan al desarrollo de redes IoT. Entre ellos, se usaría un broker para el intercambio de mensajes, y una base de datos donde almacenar los datos.
  En este despliegue, no sería necesario un script en Python para el tratado de mensajes, ya que la plataforma realiza internamente el tratado de mensajes.

  Dentro de la plataforma, se debería de:
  - *Integración del backend Loriot con Amazon AWS*: desde Loriot, se siguen los pasos para la correcta vinculación hacia AWS.
  - *Vincular dispositivos IoT en AWS*: cada dispositivo de la red de producción, debe estar registrado en la plataforma. La plataforma los denomina *Things*, y son los encargados de recibir las mediciones de los dispositivos físicos.
  - *Hacer uso del broker MQTT*: al igual que en la red de 10 dispositivos, es necesario el uso de un broker de mensajería para el intercambio de mensajes. Los dispositivos registrados en la plataforma publican las mediciones en un tópico, y el broker, suscrito al tópico, los recoge.
  - *Crear base de datos DynamoDB*: la plataforma de Amazon ofrece también la posibilidad de almacenar los datos en su base de datos, DynamoDB. Para la correcta inserción de datos, se crea una regla, que definirá que, todos los mensajes nuevos que lleguen al broker, serán insertados en las tablas de la base de datos.

  En Amazon AWS, no existen licencias. Los pagos se realizan en base a lo que se consuma de manera mensual. Estos costes, en el apartado IoT, se dividen en [37]:
  - Cargos de conectividad
  - Cargos de mensajería
  - Cargos de registro y sombra de dispositivos
  - Cargos del motor de reglas.

  Estos costes se definen en el apartado **2.3.2 Impacto económico**.

Los componentes definidos anteriormente componen la red de producción, de 10,000 dispositivos. Es un despliegue donde los componentes se encuentran más centralizados que en la red de 10 dispositivos. No obstante, podrían surgir problemas, como saturación en la red, cuellos de botella, etc., en caso de no realizar un dimensionado correcto de la misma.

4.2 Manual del usuario

El manual del usuario es un apartado donde se especifican los pasos a seguir para hacer uso de una aplicación o sistema. El manual va destinado a un usuario genérico.

En este proyecto, el manual se enfocará a la instalación y uso de la aplicación. Esto se debe a que un usuario sólo accedería a visualizar los datos en la aplicación, y no a manipular la propia red desplegada.

4.2.1 Instalación de la aplicación

La aplicación ha sido desarrollada para dispositivos móviles con sistemas operativos Android, e iOS entre otros. No obstante, estos pasos irán enfocados a la instalación sobre un dispositivo Android.

La persona encargada del desarrollo de la aplicación, debe de ser la encargada de la instalación de la aplicación sobre los dispositivos de la organización dueña de la red IoT. La razón de esto es que la aplicación no ha sido desarrollada para un público amplio, ya que los datos solamente serán visualizados y accedidos por personas relacionadas con la red.

Se dividirán los pasos en dos apartados [38].

- **Construir la APK**: para la generación de la APK, hay que acceder al código fuente de la aplicación, desde un editor de código como Visual Studio Code.
  Desde el terminal del editor:
  - Introducir cd <directorio aplicación> (Reemplazar <directorio aplicación> con el directorio de la aplicación en cuestión)
  - Ejecutar el comando flutter build apk --split-per-abi.

  Con estos simples pasos se generarán tres ficheros con la extensión .apk sobre el directorio <directorio aplicación>/build/app/outputs/apk/release/.

- **Instalar la APK en un dispositivo móvil**: para la instalación de la APK sobre un dispositivo, se seguirán los siguientes pasos, también desde la línea de comandos:
  - Conectar el dispositivo móvil al ordenador donde se encuentra el editor mediante un cable USB
  - Introducir cd <directorio aplicación>
  - Introducir flutter install.

  Con estos sencillos pasos se habrá instalado la aplicación de manera exitosa sobre nuestro dispositivo. Se podrá hacer uso pleno de la aplicación.

4.2.2 Explotación

En el apartado de explotación se definirán los pasos a seguir por un usuario para el uso correcto de la aplicación. El único elemento del proyecto que un usuario podrá usar es la aplicación, por lo que solo nos centraremos en ella.

Los pasos son los siguientes:

- Al iniciar la aplicación, el usuario visualiza la vista de logueo. En esta vista, debe de introducir mediante el teclado del dispositivo la clave correcta, almacenada en la base de datos.

Figura 68. *Vista de autenticación de la aplicación.*

  - Si el usuario introduce una clave incorrecta, la aplicación le avisará con una alerta en el centro de la pantalla del dispositivo.

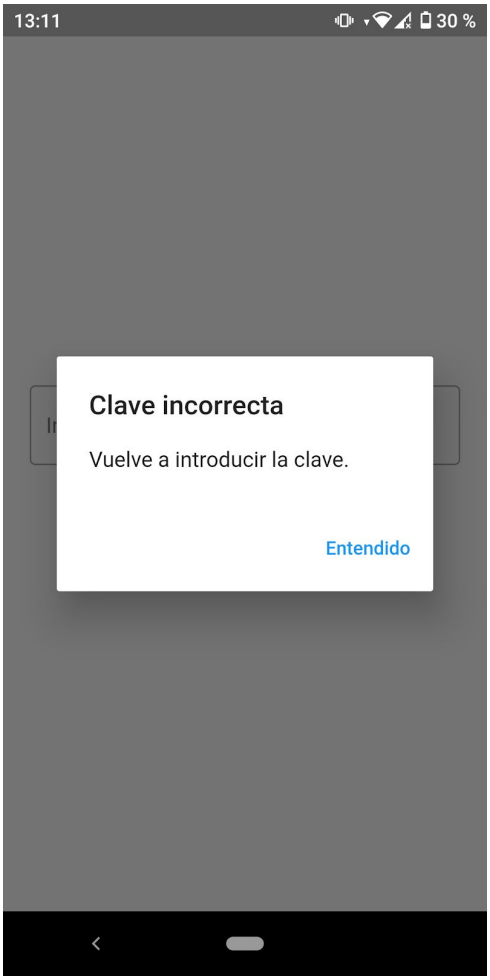


Figura 69. *Vista de autenticación fallida de la aplicación.*

- Si el usuario introduce la clave correcta, la aplicación le redirigirá a la siguiente vista.

- La siguiente vista es un menú donde el usuario selecciona el tipo de sensor del cual quiere visualizar la información. Con tan solo realizar un tap sobre una categoría, la aplicación le llevará a la siguiente vista.

Figura 70. *Vista de categorias de los sensores de la aplicación.*

- Al seleccionar una categoría, el usuario podrá visualizar las mediciones de todos los sensores de una categoría. En esta vista, se mostrarán tres tipos de datos por cada medición, que son:
    - Fecha y hora de la medición
    - Nombre o identificador de la controladora que recoge la medición
    - Valor de la medición.

  También se podrán eliminar mediciones, con presionar el icono del contenedor, a la derecha de cada medición.
  Se podrá volver a la pantalla anterior seleccionando el icono de la flecha, en la barra de navegación superior de la aplicación.

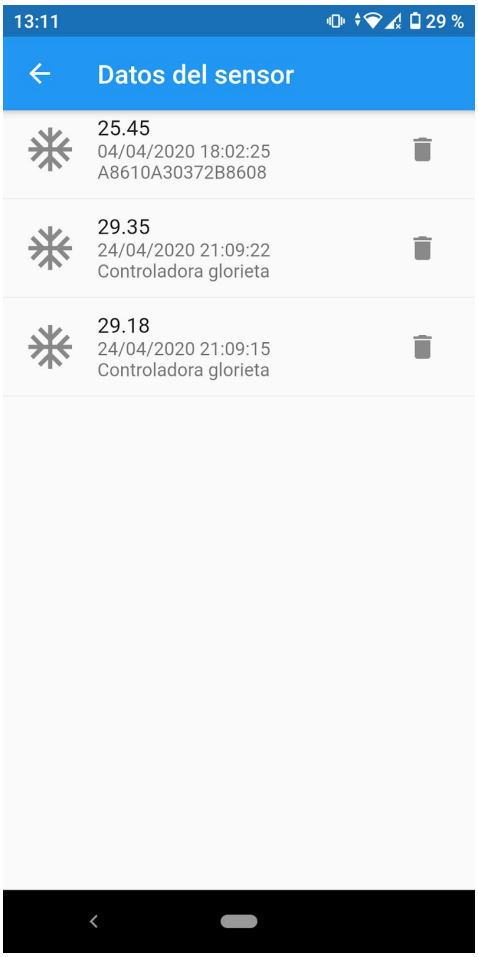


Figura 71. *Vista de mediciones de los sensores de una categoría.*

Estos son los pasos a seguir para realizar un correcto uso de la aplicación desarrollada, mostrando en ellos las distintas opciones que ofrece la misma.

5 Conclusiones

5.1 Conclusiones personales

El desarrollo del proyecto me ha servido para aprender las bases en las que se apoya una red IoT. En los inicios, al no tener conocimientos teóricos suficientes, fue complicado saber qué componentes eran necesarios para el despliegue de una red sencilla. No obstante, con la información en Internet, la aclaración de dudas a mi tutor, y las pruebas sencillas realizadas por mi mismo, el proyecto fue tomando forma.

Durante la realización del proyecto, se presentaron diversos problemas, a los que se tuvo que hacer frente de una cierta manera. O bien realizando cambios a nivel hardware en componentes, o bien cambios a nivel software entre los distintos componentes, incluyendo la sustitución de componentes.

Otro punto a destacar son las diversas maneras que existen para desarrollar un proyecto, en este caso, una red IoT. Existen muchas empresas que ofrecen servicios muy similares, algunos de ellos, con opción a licencias de uso gratuitas. En todos los componentes del proyecto, se han presentado, al menos a nivel teórico, un par de alternativas, cada una con sus características principales. Un ejemplo es el backend, que, a pesar de haberlo alojado en Loriot, habían otras opciones viables como The Things Network.
Lo necesario, al fin y al cabo, es que todos los componentes puedan integrarse de una manera correcta entre ellos, ya que todos ellos conforman la red.

La facilidad que ofrecen diversas empresas a la hora de desarrollar ciertos componentes es algo importante. En el caso del backend o la base de datos, son las propias empresas las que los crean, y simplemente dan acceso y poder de configuración al cliente que los usa. Esto es algo positivo, ya que el desarrollo de la red y sus componentes es más rápido y sencillo, permitiendo desarrollar un sistema enfocado a producción en pocos meses. Como punto negativo, se puede deducir una falta de flexibilidad en los mismos, ya que no se puede modificar la estructura de estos de una manera sencilla, y hay que adaptarse a los mismos.
En el caso de la aplicación, frameworks como Flutter hacen que el diseño de la misma, no sea un punto importante para el desarrollador, ya que es el framework el que se encarga de ofrecer módulos personalizables.

Por otro lado, destacar que la dificultad del proyecto, no ha sido en su totalidad el desarrollo y puesta en marcha de la red, sino la previsión de recursos necesarios para el correcto funcionamiento de una red, más si se enfoca a producción.
Puesto que no tengo experiencia en este tipo de proyectos, es difícil estimar el presupuesto necesario de este tipo de redes. No obstante, plataformas como Amazon AWS y Google IoT Cloud ofrecen ciertas herramientas para el cálculo del mismo.

Por último, decir que este proyecto ha sido un punto de inflexión de cara a proyectos futuros. Me ha dado la seguridad y confianza necesarios para la elaboración y el planteamiento de futuros proyectos, enfocados al IoT o a otros ámbitos.

5.2 Futuras líneas de desarrollo

Una vez realizado el proyecto, se pueden definir diversas mejoras o cambios en el mismo:

1. Mejora de la aplicación, adaptándola a las necesidades de cada red IoT. Esta se ha realizado de una manera sencilla y genérica, por lo que habría que diseñarla a medida del organismo o empresa que la use.

2. Desarrollo a nivel práctico de una red enfocada a producción. Si bien la red de 10,000 dispositivos de este proyecto es muy grande, desarrollar una con una cantidad aproximada de 100 dispositivos.

3. Realizar despliegues de sensores por distintas áreas geográficas para estudiar la cobertura de la red y sus límites. Esto sería una aproximación a una red real.

4. Resolución de algunos problemas expuestos, que por determinadas circunstancias, no han podido ser solucionados. En este caso, se podría tratar de dar solución al despliegue del script Python en una máquina virtual, sobre un sistema operativo Windows.

5. Realizar el pago de licencias de los distintos componentes de la red de 10 dispositivos. Con esto se evitarían ciertos problemas que pueden derivar del uso de licencias gratuitas, como el poco soporte que ofrecen ante problemas. Se dotaría a la red de cierta seguridad, más sabiendo que tendría recursos dedicados, y no compartidos con otros usuarios.

6 Bibliografía

1. Loriot. *LORIOT Network Server - Public Documentation (BETA) - LORIOT Network Server - LORIOT documentation*. Online

2. LoRa Alliance. *LoRaWAN 1.0.3 Specification*. Online

3. Loriot. *LORIOT - The LoRaWAN® Network Server Provider*. Online

4. The Things Network. *Network Architecture*. Online

5. The Things Network. *Applications*. Online

6. The Things Network. *Gateways*. Online

7. The Things Network. *Devices*. Online

8. Flutter. *Resumen Técnico*. Online

9. Android Studio. *Introducción a Android Studio | Desarrolladores de Android*. Online

10. Cesar Cid Robles. *Comparación bases de datos relacionales y no relacionales*. Online

11. Firebase. *Cloud Firestore | Firebase*. Online

12. MySQL. *MySQL 8.0 Reference Manual : 1.3.1 What is MySQL?*. Online

13. CloudMQTT. *Documentation*. Online

14. Amazon. *Información general sobre AWS IoT Core – Amazon Web Services*. Online

15. Sergio De Luz (2019). *Sidewalk: Tecnología inalámbrica de Amazon para dispositivos domóticos*. Online

16. Bryn Lewis (2018). *Arduino MKR WAN 1300 LoRa Field Gateway Client.* Online

17. Luigi Francesco Cerfeda (2017). *LoRaWAN Using Python, Zerynth and The Things Network*. Online

18. Sigfox. *What is Sigfox?* Online

19. Descifrando Blockchain (2018). *Cómo funciona la red LPWAN Sigfox*. Online

20. LoRa Alliance. *About LoRaWAN®*. Online

21. *Alexander La Rosa (2018). Qué es LPWAN: introducción al protocolo de comunicaciones de IoT*. Online

22. Sigfox. *Sigfox Buy*. Online

23. Lairdtech. *User Guide*. Online

24. Arduino. *MKRWAN*. Online

25. Luis Llamas. *¿Qué es MQTT? Su importancia como protocolo IoT*. Online

26. Steve (2020). *Paho Python MQTT Client - Understanding Callbacks*. Online

27. Firebase. Primeros pasos con Cloud Firestore. Online

28. Python Package Index. *paho-mqtt · PyPI*. Online

29. Firebase. *Cloud Firestore Data model | Firebase*. Online

30. Firebase. *Elige una base de datos: Cloud Firestore o Realtime Database*. Online

31. Seeed Studio. *Seeed Wiki*. Online

32. GitHub. *matthijskooijman/arduino-lmic: LoraWAN-in-C library, adapted to run under the Arduino environment*. Online

33. Mobilefish. *Build Lora node using Arduino Uno and HopeRF RFM95 LoRa transceiver module*. Online

34. Flutter. *Visual Studio Code*. Online

35. Sebastian (2019). *Getting Started With Flutter - (2) Project Structure*. Online

36. Postman. *Documenting your API*. Online

37. Amazon. *Precios de AWS IoT Core – Amazon Web Services*. Online

38. Flutter. *Build and release an Android app*. Online

7 Acrónimos

- USB: Universal Serial Bus
- P2P: Peer-to-peer
- MQTT: MQ Telemetry Transport
- SSL: Secure Sockets Layer
- TLS: Transport Layer Security
- URL: Uniform Resource Locator
- SGBD: Sistema Gestor de Base de Datos
- ACID: Atomicidad, Consistencia, Aislamiento, Durabilidad
- LPWAN: Low Power Wide Area Network
- LED: Light Emitting Diode.

8 Anexos

8.1 Codigos

Todos los códigos y ficheros empleados en este proyecto se adjuntan en el fichero comprimido **Ficheros anexos.zip**. Este comprimido contiene los siguientes ficheros:

- **Aplicación TFG**: contiene los ficheros de la aplicación desarrollada en el proyecto
- **Código Python**: contiene el proyecto desarrollado en el lenguaje Python para la captación y tratamiento de mediciones
- **Script MKR WAN 1300**: sketch de Arduino que usa el módulo LoRa MKR WAN 1300. Contiene el ejemplo que usa el sensor de temperatura
- **Script RF95**: sketch de Arduino que usa el módulo LoRa RF95. Es el módulo desarrollado a nivel teórico, por lo que el sketch no ha podido ser probado.