Universidad Zaragoza
1542

Trabajo Fin de Grado

Arquitectura Software para el despliegue masivo de dispositivos LoRaWAN

Software Architecture for the massive deployment of LoRaWAN devices

Autor/es

Francisco Javier Mínguez Carbellido

Director/es

José Félix Serna Fortea
Julio Alberto Sangüesa Escorihuela

Escuela universitaria politécnica – Teruel

2022

RESUMEN

Este proyecto consiste en el diseño y desarrollo de una arquitectura software que permita solucionar el problema que puede suponer el hecho de tener que registrar o dar de alta una gran cantidad de dispositivos LoRaWAN idénticos (mismo hardware y firmware), desplegados en localizaciones geográficas diferentes, en su respectivo sistema de gestión/explotación.

El registro de dispositivos LoRaWAN en su respectivo sistema de explotación asociado puede llegar a ser una tarea larga y costosa. Ante una posible situación en la que sea necesario un despliegue masivo de estos dispositivos, la acción de registrar cada dispositivo en su sistema de gestión puede ser un proceso tedioso de realizar, sin embargo, con la arquitectura software diseñada en este proyecto esta tarea se automatizaría. De esta forma, el proceso de dar de alta dispositivos LoRaWAN en su sistema de explotación sería más rápido y eficaz de realizar.

ABSTRACT

This project consists of designing and developing a software architecture that allows solving the problem that suppose having to register a lot of LoRaWAN devices (with the same hardware and firmware),deployed in differents geographic locations, in their respective management/exploitation system.

Registering LoRaWAN devices in their associated management system can be a long and hard task to do. Faced with a supposed situation in which a massive deployment of LoRaWAN devices is necessary, the process of registering each device in its management system can be a tedious process to carry out, however, the software architecture designed in this project can make this process automated. In this way, it is more efficient to carry out the process of registering LoRaWAN devices in their management system.

ACRONIMOS

En este documento son utilizados los siguientes acrónimos:

- **ABP**: Activation By Personalization
- **ADR**: Adaptative Data Rate
- **API**: Interfaz de Programación de Aplicaciones,
- **APP**: Apliccation
- **AppSKey**: Clave de Sesión de Aplicación
- **GPS**: Global Positioning System
- **HTTP**: HyperText Transfer Protocol
- **IoT**: Internet of Things
- **JSON**: JavaScript Object Notation
- **LoRa**: Long Range
- **LoRaWAN**: Long Range Wide Area Network
- **LPWAN**: Lower Power Wide Area Network
- **M2M**: Machine To Machine
- **MQTT**: Message Queuing Telemetry Transport
- **NwkSKey**: Clave de Sesión de Red
- **OLED**: Diodo Orgánico de Emisión de Luz
- **OTAA**: Over The Air Activation
- **TCP**: Protocolo de control de trasmsión
- **UDP**: Protocolo de datagramas de usuario
- **URL**: Uniform Resource Locator
- **USB**: Universal Serial Bus

Índice de contenido

Índice de imágenes

1. Introducción.

El mundo actual se encuentra rodeado de máquinas y objetos conectados a las redes de comunicaciones, que envían o reciben información para realizar determinadas tareas. En el momento en el que aparte de los equipos informáticos clásicos prácticamente cualquier dispositivo (relojes, lavadoras…) puede conectarse a una red y enviar o recibir información, es cuando surge el concepto de Internet of Things(IoT) [1,2].

IoT(Internet of Things) se define como la interconexión digital entre dispositivos, personas e Internet que hace posible el intercambio de datos entre ellos, permitiendo que se pueda capturar información clave sobre el uso y rendimiento de los dispositivos y objetos para detectar patrones, hacer recomendaciones, mejorar la eficiencia y crear mejores experiencias para los usuarios. Un ejemplo sencillo de IoT es la conexión entre un smartphone y los dispositivos smart que hay en una casa para controlar la iluminación o el aire acondicionado, de esta forma, desde un teléfono móvil se puede controlar el encendido y apagado del aire acondicionado o la iluminación de la casa sin necesidad de estar presente en esta [1,2].

Los dispositivos IoT utilizan la tecnología M2M (machine to machine, o máquina a máquina) en la que dos dispositivos o máquinas cualesquiera se comunican entre sí utilizando cualquier tipo de conectividad (puede ser cable, WiFi, Bluetooth, etc.), sin ser necesaria la intervención humana para realizar esta comunicación. Estos dispositivos conectados generan una gran cantidad de datos que llegan a una plataforma IoT que recolecta, procesa y analiza dichos datos. Esta información se hace relevante al usuario porque gracias a ella se pueden sacar conclusiones que le ayuden a tomar las decisiones correctas en un momento dado [1].

En los últimos años, el concepto IoT ha ganado tracción en la industria tecnológica, ya que permite que un gran número de dispositivos y sensores puedan conectarse a Internet. Las aplicaciones IoT son muy variadas, entre ellas, ciudades inteligentes, medio ambiente, agricultura, industria, seguridad, etc. Por tanto, el termino IoT es cada vez más importante en el mundo, de hecho, en nuestra vida cotidiana podemos ver una enorme cantidad de objetos conectados que forman parte del Internet de las cosas y se estima que para el año 2025 tendremos en torno a 41.600 millones de dispositivos conectados según “Worldwide Global DataSphere IoT Devices and Data Forecast” [1].

1.1 Contexto.

La gran importancia de la tecnología IoT en la actualidad y en el futuro se debe a que gracias a sus características se puede conseguir conectar numerosos sensores y dispositivos a la red, estos sensores pueden recolectar y enviar información sobre datos como la temperatura, la humedad, el CO2 o la presión atmosférica. Las aplicaciones reciben los datos enviados por los sensores, los analizan y permiten obtener conclusiones a los usuarios, así como la capacidad para poder controlar y administrar los sensores. La comunicación de los dispositivos y sensores IoT con las aplicaciones finales con las que interactúan se puede realizar utilizando diferentes tecnologías. Una tecnología que está destacando en este aspecto y que es la utilizada en este proyecto es LoRaWAN [2].

LoRaWAN es una tecnología de comunicación inalámbrica en la que los dispositivos necesitan registrarse en la red para poder comunicarse, de modo que puedan distinguirse unos de otros y ser referenciados de forma biunívoca.

En un contexto IoT (agrícola, industrial, …) en el que se requiera un despliegue masivo de dispositivos, todos ellos idénticos (mismo hardware y firmware), la acción de registrar cada uno de ellos en su aplicación o sistema de gestión/explotación correspondiente para poder interaccionar con ellos puede ser una tarea larga y complicada. Esto se debe al alto número de dispositivos empleados y a la complejidad de registrar dispositivos en una red LoRaWAN, ya que es necesario utilizar parámetros que son difíciles de configurar y obtener en los dispositivos para que puedan ser identificados de forma biunívoca dentro de la red LoRaWAN. Esta situación provoca que registrar dispositivos en una red LoRaWAN, sobre todo si es una gran cantidad de ellos, pueda llegar a suponer un problema difícil de abordar.

1.2 Objetivos.

Ante la situación expuesta en la sección anterior (1.1 Contexto), se han definido los siguientes objetivos:

- Diseñar y desarrollar una arquitectura Software que presente una solución al problema de registrar la identidad única y localización de cada dispositivo en su sistema de gestión/explotación asociado.

- Demostrar que la arquitectura software propuesta pueda cumplir con el objetivo para el cual ha sido diseñada, facilitando así el registro de los dispositivos en su respectivo sistema de gestión.

2. Estado de la técnica.

Esta sección recoge un análisis sobre trabajos que ya han sido realizados previamente y contienen unos objetivos similares a los propuestos en este proyecto.

Algunos artículos encontrados recogen la importancia de los despliegues de grandes cantidades de dispositivos IoT para diferentes ámbitos y otros hablan sobre las tecnologías actuales que facilitan el despliegue masivo de dispositivos IoT, como la tecnología que ha sido utilizada en este proyecto, LoRaWAN. Pese a todo no se ha encontrado ningún artículo o trabajo que detalle con precisión como realizar el proceso para registrar una gran cantidad de dispositivos IoT en su entorno de gestión asociado, lo cual es el objetivo principal en la realización de este proyecto.

Finalmente, los trabajos encontrados que se van a destacar en esta sección por ser de utilidad en este proyecto son:

- Un blog, escrito por Jayna Locke (directora de Marketing en Digi):

  Este blog destaca la importancia de tener una plataforma de gestión de dispositivos IoT, para obtener información sobre el rendimiento de todos los dispositivos, solucionar problemas y realizar actualizaciones. Además, expone varios ejemplos en los que se ha utilizado un despliegue de numerosos dispositivos IoT (agricultura, gestión del tráfico, iluminación inteligente en ciudades…) e indica las funciones y características de las que debe disponer una plataforma de gestión de dispositivos IoT.

  - Enlace: Blog Jayna Locke

- Un trabajo de Master, escrito por Michael Andrés Moya Quimbita:

  Este trabajo explica cómo funciona la tecnología LoRAWAN y analiza su cobertura en un entorno urbano como es la ciudad de Valencia. Se puede ver la diferencia entre usar la tecnología en un entorno urbano con objetos de por medio que dificultan la conectividad (rango de cobertura 55-100 metros) y un entorno rural al aire libre (rango de cobertura alrededor de 2km).

  - Enlace: Trabajo Master Michael Andre Moya Quimbita

- Un Trabajo Fin de Grado, escrito por Marina Marín Cava:

  Este trabajo explica en detalle las características de la tecnología LoRaWAN e implementa un gateway LoRa y un nodo móvil con sensores. El nodo móvil se va desplazando por determinadas zonas desde las que se disponga de cobertura y envía datos a su aplicación asociada como la temperatura, el CO2 o la ubicación.

  - Enlace: TFG Marina Marín Cava

3. Tecnologías empleadas.

Ante la necesidad de conectar dispositivos en entornos IoT, aparecen numerosas soluciones tecnológicas, algunas son popularmente conocidas como los casos de Bluetooth o WiFi, sin embargo, estas tecnologías tienen un corto alcance y requieren que los dispositivos se encuentren relativamente cerca entre ellos para disponer de conectividad. Esto dificulta que sean empleadas en entornos IoT.

La tecnología que será utilizada para la conexión de los dispositivos en este proyecto debido a su popularidad en entornos IoT y sus características es LoRaWAN.

3.1 LoRaWAN

LoRaWAN es un sistema de red que está diseñado para establecer comunicaciones inalámbricas de largo alcance entre objetos o dispositivos de baja potencia. Sus frecuencias de trabajo están en la banda de 868 Mhz en Europa, 915 Mhz en América y 433 Mhz en Asia [6].
Las principales características de esta tecnología son:

- **Área grande de cobertura**: Permite el transporte inalámbrico de datos entre dispositivos separados por distancias kilométricas.
- **Bajo consumo**: Las baterías de los dispositivos suelen durar grandes periodos de tiempo, alcanzando en algunos casos los 7 años.
- **Baja velocidad de transmisión**: Las velocidades de datos se encuentran en el rango de 0.3 kbps a 50 kbps.
- **Cantidad de datos transmitidos**: El tamaño de los paquetes de datos enviados es pequeño, con una máximo de 256 Bytes.
- **Comunicación bidireccional**: Se permite la comunicación bidireccional, aunque en la mayoría de los casos lo habitual será que el nodo final mande datos al servidor.
- **Administración de la tasa de datos**: LoRaWAN es capaz de administrar la tasa de datos y la potencia de salida de radio frecuencia para cada uno de los dispositivos de la red, utilizando un esquema adaptativo de velocidad de datos (ADR) para maximizar la duración de la batería de los dispositivos y la capacidad de la red. [5,6]

Debito a estas características LoRaWAN es una red idónea en el ámbito IoT, ya que los dispositivos y sensores que se conectan, en la mayoría de los casos, a largas distancias, transmiten poco volumen de información y, a veces, no de forma constante en el tiempo.

3.1.1 Arquitectura de una red LoRaWAN

La arquitectura de una red LoRaWAN tiene una topología en estrella, en la que los dispositivos finales establecen directamente comunicación con un Gateway, el cual retransmite los mensajes entre los nodos finales y el servidor de red. Es posible pensar en aplicar una topología malla, en la que los dispositivos finales reenvían los mensajes de otros dispositivos con el fin de aumentar la cobertura, pero esto para aplicaciones dependientes de baterías no resulta la mejor opción, debido al elevado consumo energético que tendría cada dispositivo por estar constantemente escuchando y reenviando datos que en realidad son irrelevantes para ellos [8].

En una red LoRaWAN, los nodos finales no están asociados con un gateway específico, por lo cual, los datos transmitidos por un nodo pueden llegar a ser recibidos por todos los gateways que estén dentro de su alcance. Cada gateway reenviará el paquete recibido desde el nodo final al servidor de red, a través de tecnologías como Ethernet o WiFi [8,9].

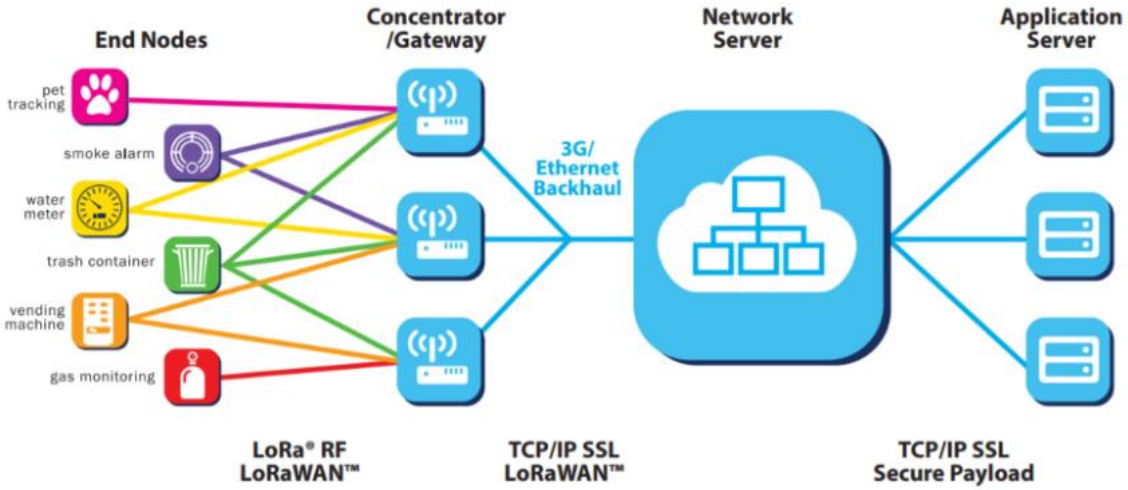

Figura 1. Arquitectura red LoRaWAN [6].

Como se ve en la imagen expuesta los elementos que componen esta arquitectura son:

- **Nodos finales**: Son dispositivos que están conectados de forma inalámbrica a una red LoRaWAN a través de gateways LoRa, utilizando la modulación de RF LoRa. Normalmente, un nodo final es un sensor autónomo o un actuador, a menudo operado por batería, cuya función es obtener información ambiental (temperatura, humedad, …) y enviarla al Gateway, o bien, estar a la espera de una orden para realizar una acción. Algunos ejemplos de actuadores son: alumbrado público, cerraduras inalámbricas, control de válvulas de agua [9].

  Cuando se fabrican los nodos finales LoRa se les asignan varios identificadores únicos. Estos identificadores se utilizan para activar y administrar de forma segura el dispositivo, garantizar el transporte seguro de paquetes a través de una red pública o privada y para entregar datos cifrados a la nube [9].

- **Gateways**: Son estaciones base implementadas por LoRa, las cuales reciben información de múltiples dispositivos finales y se encargan de reenviarla a los servidores de red. Cada gateway reenviará los paquetes recibidos desde el nodo final al servidor de red a través de conexiones IP estándar [9].

- **Servidores de red**: Se encargan de gestionar toda la red, controlando los parámetros de esta para adaptar el sistema a las condiciones siempre cambiantes, y establecer conexiones seguras para el transporte de datos extremo a extremo. El servidor de red asegura la autenticidad de cada sensor en la red y la integridad de cada mensaje. Concretamente, el servidor de red se encarga de comprobar la dirección del dispositivo, autenticar la trama, gestionar el contador de tramas, eliminar mensajes duplicados, adaptar las velocidades de datos mediante el protocolo ADR, reenviar cargas útiles de mensajes uplink a los servidores de aplicaciones apropiados, poner en cola las cargas útiles de mensajes downlink que provienen de cualquier servidor de aplicaciones conectado a la red …[9].

- **Servidores de aplicaciones**: Los servidores de aplicaciones son responsables de manejar, gestionar e interpretar de forma segura los datos enviados desde los dispositivos. También generan las cargas útiles de enlace descendente en los mensajes enviados desde la capa de aplicación a los nodos finales conectados [9].

3.1.2 Mensajes de comunicación

En las redes LoRaWAN se pueden distinguir los siguientes mensajes:

- **Mensajes uplink (enlace ascendente)**: Estos mensajes se envían desde los dispositivos finales hacia el servidor de red atravesando uno o varios gateways. Los mensajes de enlace ascendente están formados por un preámbulo, un encabezado físico (PHDR) y la carga útil (PHYPayload), estos dos últimos seguidos con su CRC correspondiente (PHDR_CRC y CRC) que se encargan de asegurar una correcta transmisión [5].

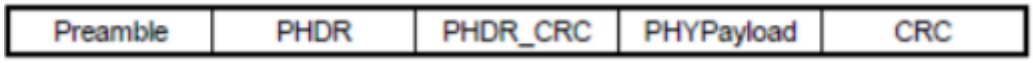

Figura 2. Formato mensaje uplink [5].

- **Mensajes downlink (enlace descendente)**: Los mensajes de enlace descendente se envían desde el servidor de red hacia un solo dispositivo final, atravesando un único gateway. En este caso llevan preámbulo, encabezamiento y payload pero únicamente tienen CRC en el encabezamiento [5].

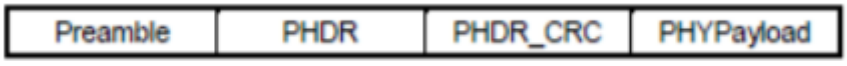

Figura 3. Formato mensaje downlink [5].

3.1.3 Clases de dispositivos finales

Los dispositivos finales basados en LoRa se clasifican en tres opciones Clase A, Clase B y Clase C dependiendo de la forma en la que estos se comunican con la red. Todos los dispositivos deben admitir el funcionamiento de Clase A. Los dispositivos de Clase B deben admitir los modos de Clase A y Clase B, y los dispositivos de Clase C deben admitir los tres modos de operación.

- **Clase A**: Son dispositivos finales bidireccionales que pasan la mayor parte del tiempo en un estado inactivo. Los dispositivos finales, realizan la transmisión de un paquete (Uplink) y acto seguido abren dos ventanas de recepción (Rx1 y Rx2) de mensajes (Downlink) para aceptar una confirmación del paquete enviado o aceptar otros datos enviados a través del gateway. Las ventanas de recepción deben durar como mínimo lo suficiente para detectar el preámbulo de un mensaje Downlink, y si este se detecta, el dispositivo se queda escuchando hasta recibir toda la trama completa. El proceso que se realiza es el siguiente:

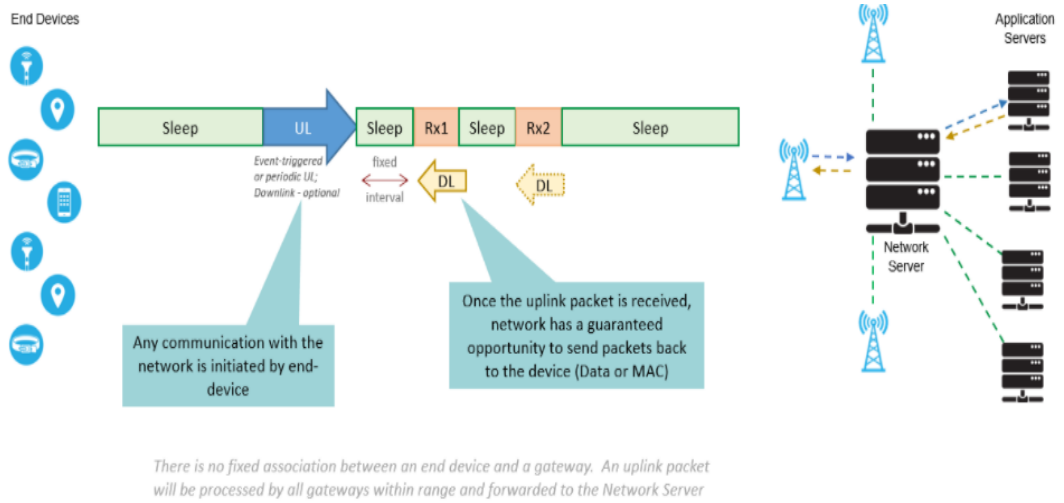

Figura 4. Comunicación de dispositivos Clase A [8].

El dispositivo final se despierta cada cierto tiempo, según este programado, y envía un mensaje uplink, después abre la primera ventana de recepción (Rx1) generalmente durante un segundo (aunque esta duración es configurable). En este punto pueden ocurrir dos situaciones:

- Se recibe un mensaje Downlink: En este caso el dispositivo actúa ante el mensaje recibido y vuelve a entrar en modo suspensión hasta que vuelva a transmitir datos.

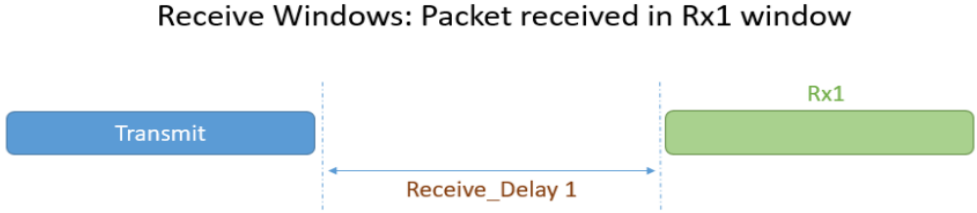

Figura 5. Paquete downlink recibido en Rx1 [8].

- No se recibe nada: En esta situación el dispositivo entra en suspensión durante un tiempo muy breve y después abre la segunda ventana de recepción (Rx2). En esta segunda ventana si no se recibe ningún mensaje el dispositivo entra en suspensión hasta que vuelva a transmitir. En caso contrario, si se recibe algún mensaje, se actúa ante este y después se entra en modo suspensión.

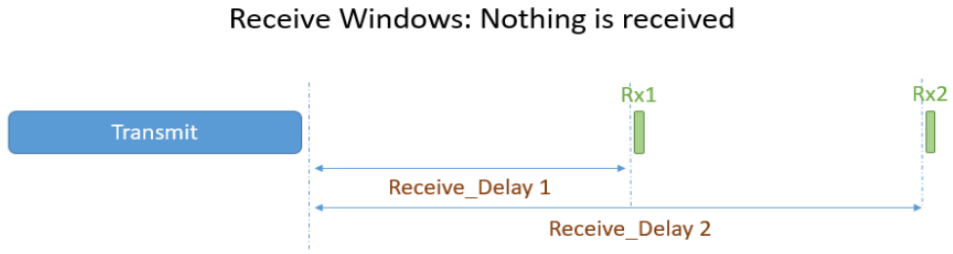

Figura 6. El nodo final no recibe ningún paquete downlink [8].

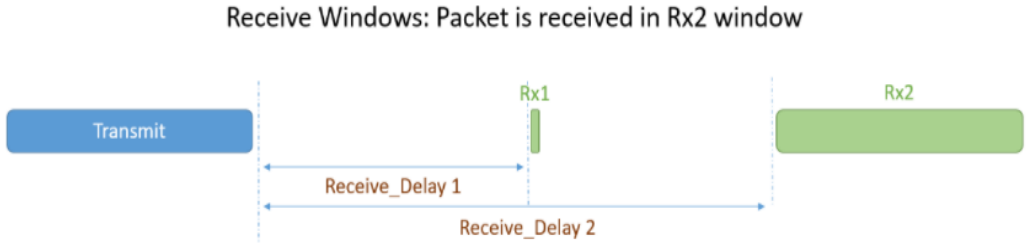

Figura 7. Paquete downlink recibido en Rx2 [8].

Hay que destacar que no hay forma de que una aplicación exterior asociada a un dispositivo de Clase A pueda reactivar este. Dada esta limitación, los dispositivos de Clase A no son adecuados para actuadores, sin embargo, al estar la mayor parte del tiempo en un estado de suspensión son los más eficientes energéticamente. Esto les hace idóneos en entornos en los que apenas sean necesarios mensajes downlink y el tiempo de diferencia entre los mensajes de transmisión sea elevado [8,9].

- **Clase B**: Los dispositivos de Clase B permiten que la aplicación les envíe datos de forma programada. Para conseguir una sincronización del Gateway con el nodo final, se envían tramas *beacon*. Estas tramas son señales periódicas que envía el gatewey al nodo final para que este pueda alinear sus relojes internos con la red. De esta forma el servidor red conoce el momento en el que el dispositivo final está escuchando. Por tanto, los dispositivos finales pueden abrir ventanas de recepción periódicamente y cualquiera de estas ranuras puede ser utilizada por la infraestructura de red para enviar un mensaje downlink al dispositivo. Estos dispositivos tienen un mayor gasto energético que los de Clase A [8,9].

- **Clase C**: Los dispositivos de clase C siempre está "activos". Estos dispositivos siempre están escuchando mensajes downlink, a menos que se encuentren

transmitiendo un mensaje uplink. Los dispositivos finales de Clase C implementan las mismas dos ventanas de recepción que los dispositivos de Clase A, pero no cierran la segunda ventana (Rx2) hasta que envían la siguiente transmisión al servidor. Por lo tanto, pueden recibir un mensaje downlink en la ventana Rx2 en casi cualquier momento. Estos dispositivos consumen una gran cantidad de energía, por lo que se recomienda su uso cuando cuenten con una fuente de alimentación externa. [8,9]

Esquema de latencia y duración de los dispositivos según su clase:

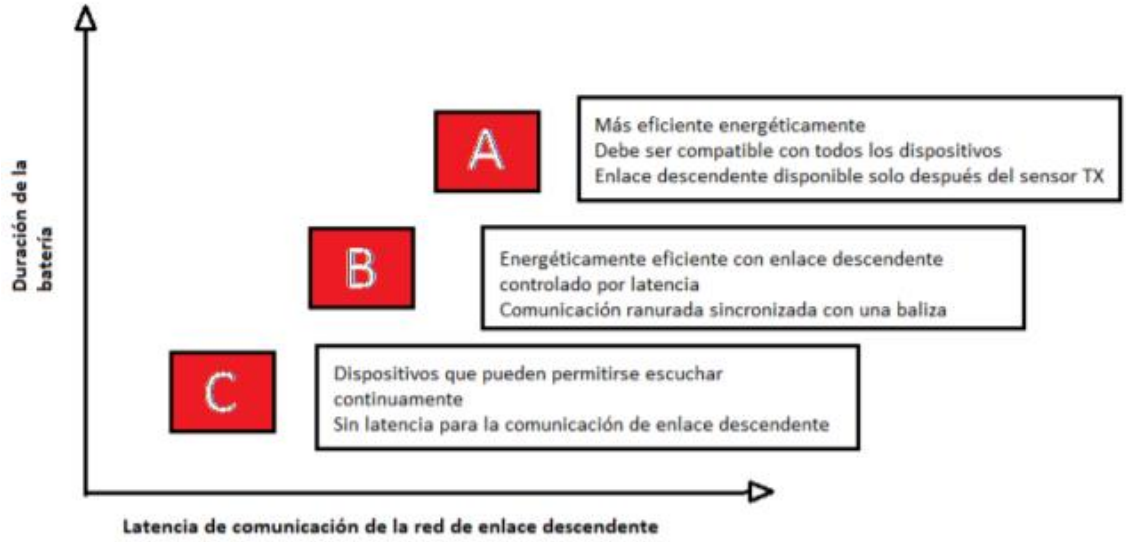

Figura 8. Comparación de clases de dispositivos [12].

3.2 Chirpstack

Chripstack es un sistema basado en software libre que proporciona la capacidad de crear y gestionar redes LoRaWAN, normalmente la redes suelen ser privadas [11,12].

Permite conectar dispositivos finales LoRa y gateways LoRaWAN asociándolos a un sistema para poder obtener comunicación con estos, ya que proporciona una interfaz de usuario amigable que permite gestionar dispositivos, usuarios, aplicaciones, organizaciones y gateways. Además, dispone de una API que facilita su integración con aplicaciones externas de terceros [11,13].

El principal objetivo que persigue ChirpStack es proporcionar la infraestructura necesaria para recibir información de dispositivos y gateways LoRa, con el fin de dar capacidades de gestión de dichos dispositivos (por un lado) y de poner la información que envían los dispositivos finales a disposición de sistemas externos para que la consuman [10].

La arquitectura de una red gestionada por Chirpstack sigue básicamente la estructura de una red LoRaWAN (3.1.1 Arquitectura de una red LoRaWAN), por tanto, esta se articula en los siguientes elementos:

- **Dispositivos finales LoRa:** Dispositivos de campo que envían por tecnología LoRa información de los sistemas que controlan (sensor de temperatura, el propio estado del dispositivo, etc…) a un gateway, o bien que reciben información de este Gateway para realizar una acción (activar un relé, encender un led…) [10].

- **Gateway LoRaWAN**: Elemento que recibe información de los dispositivos, y transforma un paquete LoRa en un paquete IP (bien TCP o UDP, aunque lo más común es lo primero), transfiriendo la información que proporciona el dispositivo hacia un servidor donde esta información es procesada. También tiene capacidad de enviar información o solicitud de acciones a los dispositivos por parte de este servidor. Junto con los dispositivos LoRa, constituyen los elementos de campo, y aunque no forman parte estrictamente hablando de ChirpStack, sí tienen una interacción muy cercana con él [10].

- **Gateway Bridge**: Es el primero de los componentes de ChirpStack, si seguimos el flujo de datos desde los dispositivos de campo hasta los servidores de computación. Su función es recibir la información de los gateways y procesarla, volcándola en un servidor MQTT de mensajería. Este bridge puede residir en el servidor donde se despliegue ChirpStack, en los propios gateways LoRa o estar instalado en un tercer componente aparte. Su función primordial, en pocas palabras, es volcar la información proveniente de la red LoRaWAN en el sistema de mensajería MQTT, donde será consumida por el resto de los servicios de ChirpStack [10].

- **Network Server**: Segundo de los componentes de ChirpStack. Es el servidor de red LoRaWAN propiamente dicho. Se encarga de monitorizar el estado de la red, los dispositivos conectados a la misma, y administrar el acceso de nuevos dispositivos a la red. También se encarga, en el caso de redes con múltiples gateways, de resolver los mensajes duplicados (dado que un paquete enviado por un dispositivo puede ser recibido y procesado por más de un Gateway), consolidar la información, y ponerla a disposición del servidor de aplicaciones de ChirpStack. También se encarga de las siguientes funcionalidades: Autenticación de dispositivos, gestión de la capa mac LoRaWAN, gestionar el envío de mensajes desde ChirpStack a los dispositivos [10].

- **Application Server**: Tercer componente de ChirpStack. Es la parte visible de la arquitectura ya que proporciona la interfaz web. Permite al usuario gestionar

dispositivos, gateways, usuarios, crear "aplicaciones", que en este contexto son grupos de dispositivos que envían una información del mismo tipo. Relaciona la información enviada por uno o varios dispositivos, almacenando un histórico, y la pone a disposición de terceros sistemas mediante diversos métodos de integración [10].

- **Geolocation server**: Componente opcional que permite dotar de mayores capacidades de geolocalización a los dispositivos, en caso de que el Gateway no proporcione esta información, o en el que se quiera hacer un tratamiento personalizado de la misma [10].

- **Broker MQTT**: Utilizado como sistema de mensajería interna para el resto de las componentes de ChirpStack y la comunicación con los gateways [10].

- **Redis**: Motor de base de datos en memoria, que gestiona la información que se intercambia entre los dispositivos y aplicaciones creadas en ChirpStack [10].

- **Base de datos PostgreSQL**: Almacena información de configuración de ChirpStack, organizaciones, aplicaciones, usuarios, etc… además de información histórica enviada por los dispositivos. Existen diversos mecanismos (HTTP, MQTT, InfluxDB, RabbitMQ, PostgreSQL, API REST) [10].

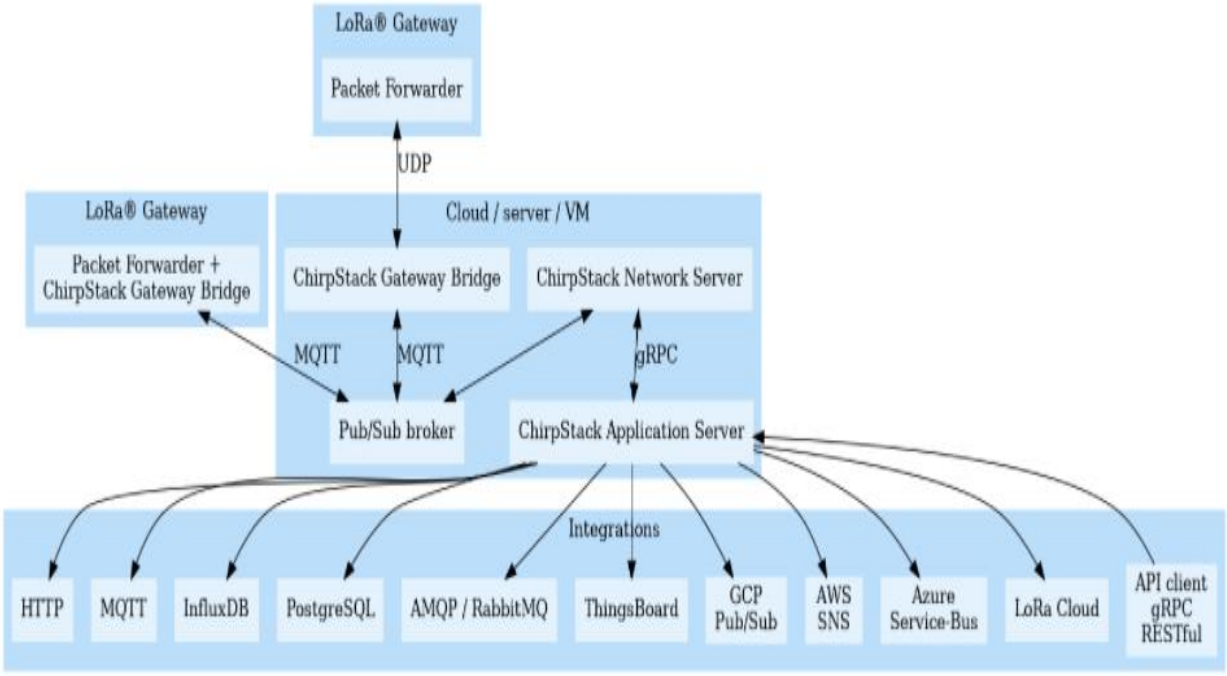

Figura 9. Arquitectura Chirpstack [12].

Un aspecto interesante es que Chirpstack se puede desplegar de diferentes maneras, al estar estructurado en una serie de componentes bien definidos que se comunican entre ellos mediante puertos e interfaces estandarizados. Por lo tanto, permite tanto realizar un despliegue convencional en un único servidor, como desplegarse en un modelo de microservicios en un entorno Docker o Kubernetes [10, 11].

También hay que destacar que ChirpStack hace uso de los componentes anteriores para componer y almacenar información proveniente de los dispositivos de campo de una forma estructurada, siguiendo un formato JSON. Esto beneficia un fácil análisis y procesamiento de la información recibida a través de los nodos finales. Un ejemplo de mensaje enviado desde un nodo final es el siguiente:

```
{
   “applicationID”: “123″,
   “applicationName”: “temperature-sensor”,
   “deviceName”: “garden-sensor”,
   “devEUI”: “0202020202020202″,
   “rxInfo”: [
              {
               "gatewayID": "0303030303030303",
               "name": "rooftop-gateway",
               "time": "2016-11-25T16:24:37.295915988Z",
               "rssi": -57,
               "loRaSNR": 10,
               "location": {
                        "latitude": 52.3740364,
                        "longitude": 4.9144401,
                        "altitude": 10.5
               }
              }
            ],
  “txInfo”: {
            “frequency”: 868100000,
            “dr”: 5
   },
  “adr”: false,
  “fCnt”: 10,
  “fPort”: 5,
  “data”: “…”,
  “object”: {
            “temperatureSensor”: {“1″: 25},
            “humiditySensor”: {“1″: 32}
   },
  “tags”: {
            “key”: “value”
   }
}
```

A continuación, se muestran algunas imágenes sobre la interfaz web de Chirpstack:

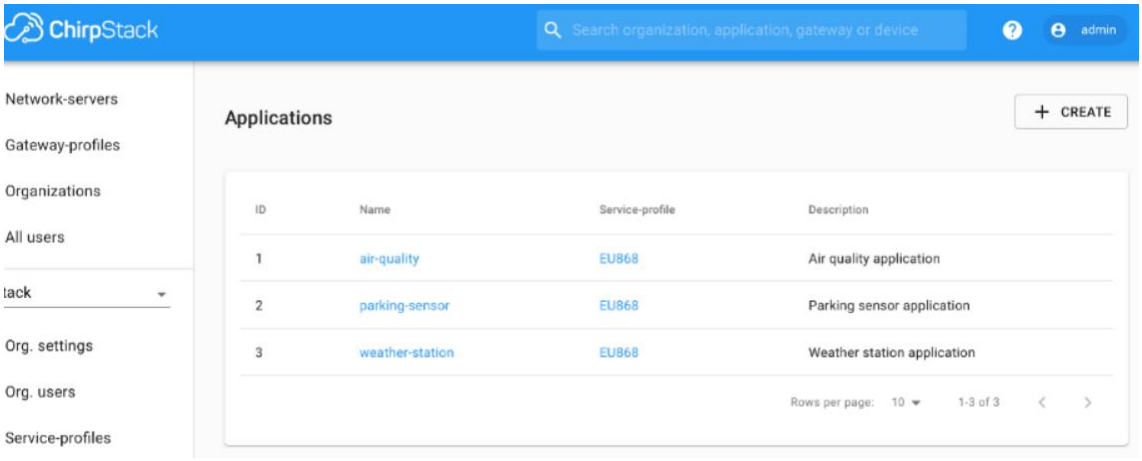

Figura 10. Interfaz web Chirpstack, aplicaciones registradas [10].

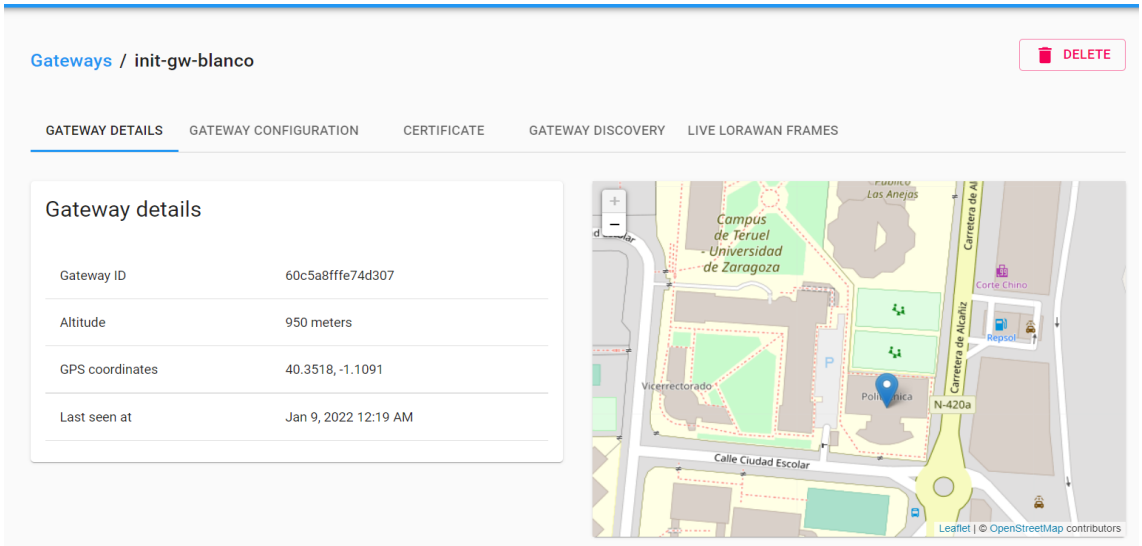

Figura 11. Interfaz web Chirpstack, información sobre un Gateway.

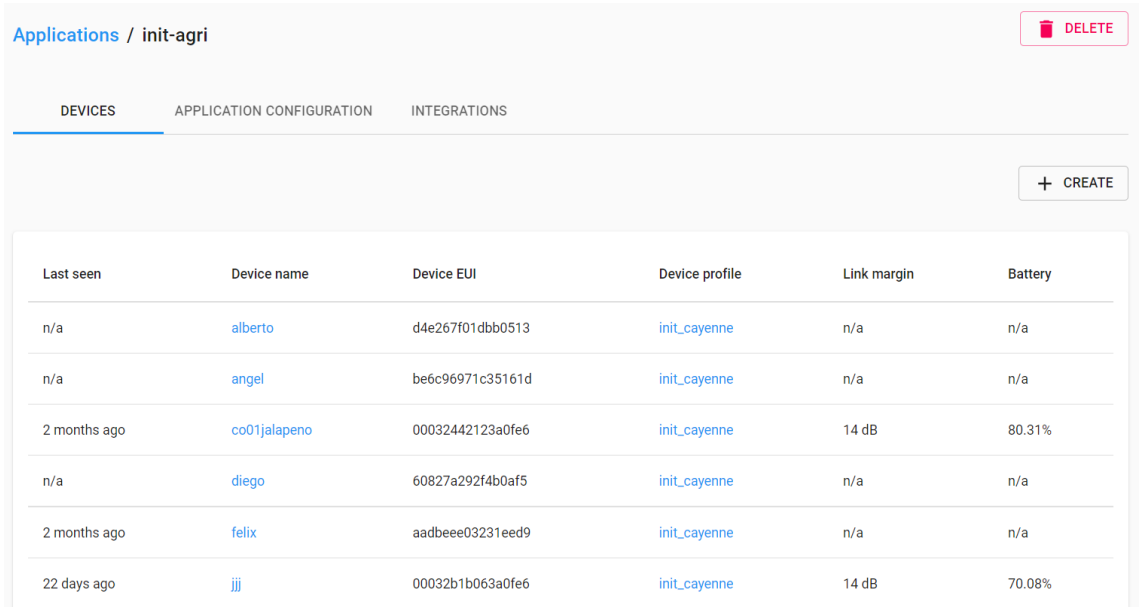

Figura 12. Interfaz web Chirpstack, dispositivos registrados y asociados a una aplicación.

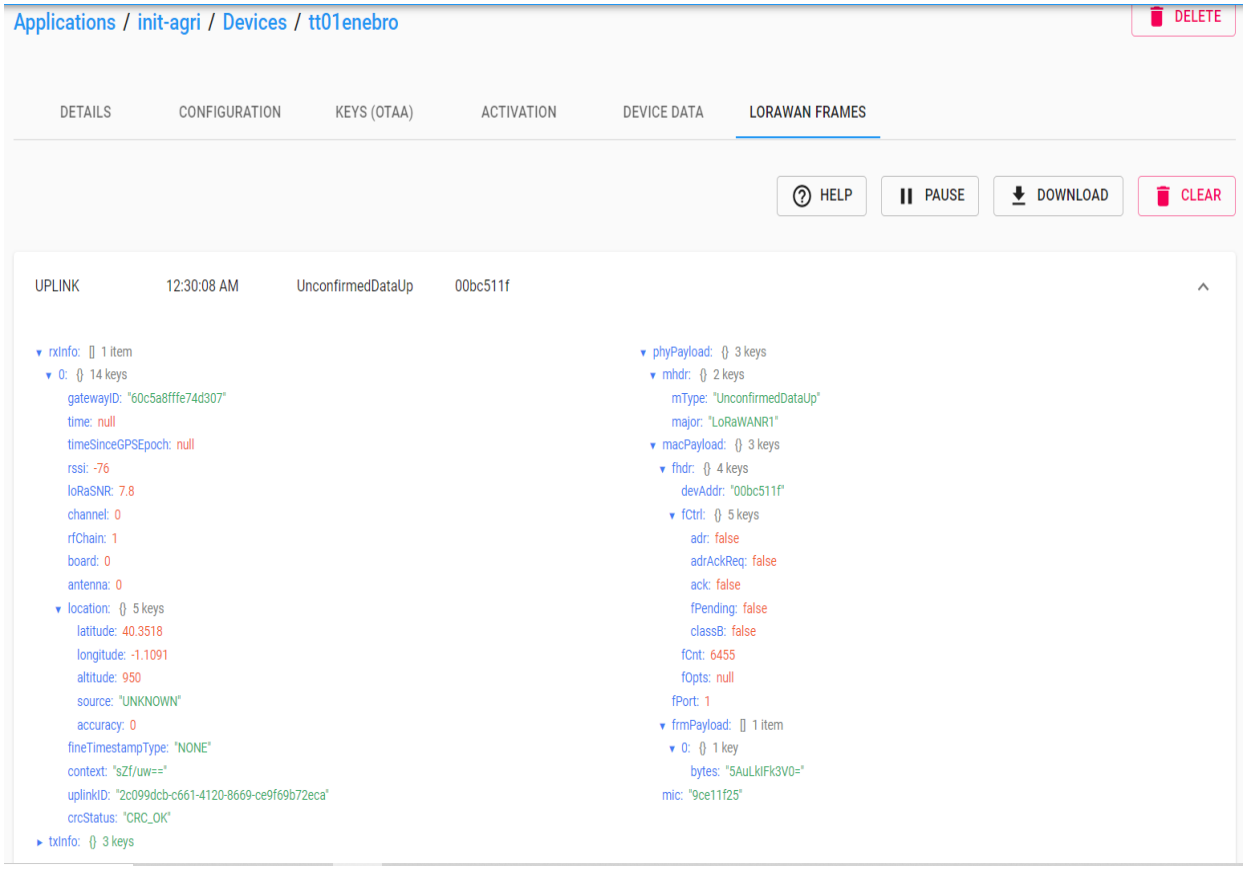

Figura 13. Interfaz web Chirpstack, mensaje uplink enviado por un dispositivo.

4. Materiales hardware

Esta sección recoge los elementos hardware que se han utilizado en este proyecto, dando una breve descripción sobre sus principales características y funciones.

4.1 Módulo Cubecell HTTC-AB02

HelTec CubeCell es una nueva serie de productos fabricados por HelTec para el desarrollo de aplicaciones del Internet de las Cosas mediante nodos LoRa/LoRaWAN.
La serie CubeCell (TM) se basa en ASR605x (ASR6501, ASR6502), esos chips ya están integrados con el MCU de la serie PSoC 4000 (ARM Cortex M0 + Core) e incluyen el transceptor LoRa de Semtech SX1262 [4].

El módulo HTTC-AB02 (utilizado en este proyecto) es un dispositivo perteneciente a la Clase A respecto a su comunicación con la red. Este permite la agregación de diferentes sensores a él para recolectar información (temperatura, presión, etc.), posteriormente estos datos pueden ser transmitidos a través de una red LoRaWAN [14].

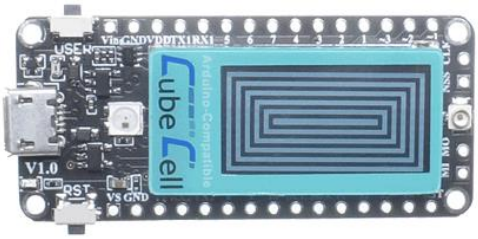

Figura14.Módulo Cubecell AB02 [14].

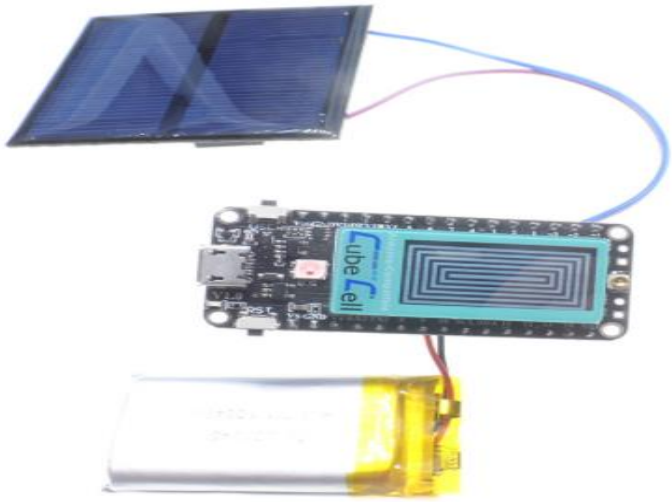

Figura 15. Módulo AB02 con batería panel solar externa [14].

Las características de este dispositivo son:

- Compatibilidad con la tecnología Arduino.
- Compatibilidad con la tecnología LoRaWAN.
- Diseño de potencia ultra bajo, 3,5uA en modo suspensión.
- Disposición de un sistema de gestión de energía solar, el sistema se puede conectar directamente con un panel solar de 5,5 ~ 7 V. (Figura 15)

- Interfaz de batería SH1.25-2 integrada, sistema de gestión de batería de litio integrado (gestión de carga y descarga, protección contra sobrecarga, detección de energía de la batería, conmutación automática de energía entre batería y USB).
- Interfaz micro USB con protección ESD completa, protección contra cortocircuitos, blindaje RF y otras medidas de protección.
- Chip integrado CP2102 USB a puerto serie, conveniente para la descarga de programas, impresión de información de depuración.
- Buena correspondencia de impedancia y larga distancia de comunicación.
- Pantalla OLED (128 * 64 pixeles) 0,96 pulgadas integrada, que se puede usar para mostrar información de depuración, energía de la batería y otra información [14].

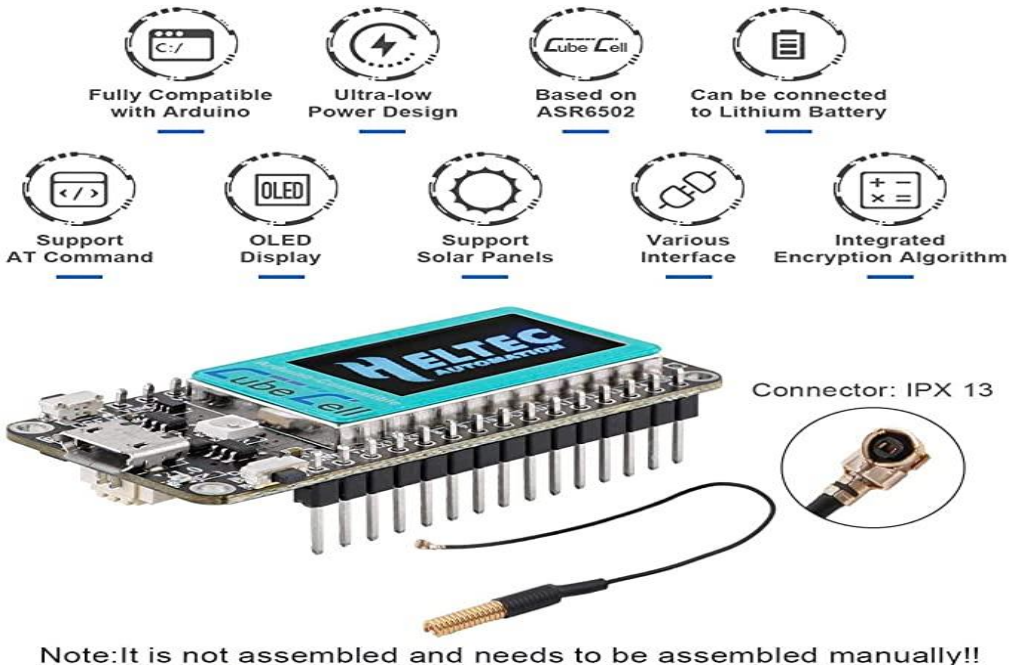

Figura 16. Características módulo AB02 [14].

Algunos parámetros técnicos y eléctricos del dispositivo como la capacidad de la memoria flash, el tipo de batería, etc… se muestran en las siguientes imágenes.

Parámetros técnicos del dispositivo

Resource	Parameter	
Master Chip	ASR6502 (48 MHz ARM® Cortex® M0+ MCU)	
Wireless Communication	LoRa	
	Node-to-node communication or LoRaWAN	
LoRa Chip	SX1262	
LoRaWAN Area	hardware version	Support frequency
	LF	EU433
		CN470
	HF	IN865
		EU868
		US915
		AU915
		KR920
		AS923
LoRa Maximum Output Power	22dB ± 1dB	
Hardware Resource	UART x 2; SPI x 2; I2C x 2; SWD x 1; 12-bits ADC input x 3; 8–channel DMA engine; GPIO x 16	
FLASH	128KB internal FLASH	
RAM	16KB internal SRAM	
Interface	Micro USB x 1; LoRa Antenna interface(IPEX) x 1; 15 x 2.54 pin x 2+2 x 2.54 pin x 3	
Maximum Size (Including protruding parts such as switch and battery compartment)	51.9 x 25 x 8 mm	
USB to Serial Chip	CP2102	
Battery	3.7V Lithium (SH1.25 x 2 socket)	
Solar Energy	5.5 ~ 7V solar panel	
Battery Detection Circuit	√	
External Device Power Control (Vext)	√	
Low Power	Deep Sleep 3.5μA	
Display Size	0.96-inch OLED	
Working Temperature	-40~80℃	

Figura 17. Parámetros técnicos módulo AB02 [14].

Parámetros eléctricos del dispositivo

Electrical Features	Condition	Minimum	Typica	Maximum
Power Supply	USB powered (≥500mA)	4.7V	5V	6V
	Lithium powered (≥250mA)	3.3V	3.7V	4.2V
	3.3V (pin) powered (≥150mA)	2.7V	3.3V	3.5V
	5V (pin) powered (≥500mA)	4.7V	5V	6V
Power Consumption(mA)	LoRa Rx Mode		10mA	
	LoRa 10dB output		70mA	
	LoRa 14dB output		90mA	
	LoRa 17dB output		100mA	
	LoRa 20dB output		105mA	
	Sleep Mode (USB powered)		9.6mA	
	Sleep Mode (VBAT/battery powered)		11μA	
	Sleep Mode (3.3V header powered)		3.5μA	
Output	3.3V pin output			500mA
	5V pin output (USB powered only)		Equal to the input current	
	External device power control (Vext 3.3V)			350mA

Figura 18. Parámetros eléctricos módulo AB02 [14].

4.2 Gateway LoRa

El Gateway que se ha empleado en este proyecto como parte de la red LoRaWAN es el modelo *WisGate Edge Lite RAK7258*, en su variante no LTE. *WisGate Edge Lite RAK7258* es una puerta de enlace LoRa de 8 canales con conectividad Ethernet y que dispone de configuración Wi-Fi integrada que le permite configurarse a través del modo AP Wi-Fi predeterminado [18].

El software de código abierto para la gestión y configuración de este dispositivo Gateway se basa en OpenWRT. Tiene un reenviador de paquetes LoRa incorporado y una interfaz gráfica de usuario, lo que permite una configuración rápida sin renunciar a la libertad de una solución totalmente personalizada [18].

El Gateway tiene un rango de distancia de hasta 15 km, mientras que para entornos altamente urbanizados puede cubrir más de 2 km y es compatible con el modo MQTT Bridge lo que le permite integrarse fácilmente en una red LoRaWAN gestionada a través de Chirpstack haciéndolo idóneo para este proyecto [18].

Figura 19. Gateway Lora WisGate Edge Lite RAK7258 [18].

5. Desarrollo del proyecto

Conociendo en profundidad las tecnologías software que se van a emplear y una vez conseguidos los materiales hardware que anteriormente han sido comentados, se dispone de la siguiente infraestructura hardware/software para desarrollar el proyecto:

- Servidor ejecutando Chirpstack.
- Servidor Ubuntu 20.04.1.
- Gateways LoRaWAN.
- Dispositivos cubecell AB02.

Teniendo en cuenta la infraestructura hardware/software de la que se dispone y los objetivos a conseguir, el desarrollo de la solución que propone este TFG se ha estructurado en las siguientes actividades:

- Registro de un dispositivo final en una red LoRaWAN (Procedimiento JOIN).
- Diseño de la arquitectura software.
- Desarrollo de Chirpcontrol (APP web externa).
- Desarrollo del firmware del dispositivo final.
- Desarrollo de ChirpRegister (APP Android).

5.1 Registro de un dispositivo final en una red LoRaWAN (JOIN).

En este apartado se ha hecho un estudio sobre todo el proceso que se realiza cuando un dispositivo tiene que ser dado de alta en una red LoRaWAN (procedimiento JOIN). La acción de dar de alta un dispositivo en su sistema de gestión correspondiente es el centro de este proyecto, ya que el objetivo que se persigue es poder realizar este proceso de una manera más eficaz y automatizada.

El procedimiento JOIN establece la autenticación mutua entre un dispositivo final y la red LoRaWAN a la que está conectado. De esta forma solo los dispositivos autorizados pueden unirse a la red y se garantiza que los mensajes intercambiados entre los dispositivos finales y su respectiva aplicación de gestión estén autenticados por origen, protegidos por integridad y encriptados de extremo a extremo (es decir, desde el dispositivo final hasta el servidor de aplicaciones y viceversa) [8].

Como ya se ha comentado con brevedad anteriormente, los dispositivos finales LoRa suelen tener unos parámetros únicos que les diferencian del resto de dispositivos, estos parámetros están diseñados para que un dispositivo pueda establecer conexión con una red LoRaWAN y su función es garantizar que el procedimiento JOIN se pueda cumplir de forma correcta en términos de seguridad. Algunos parámetros los puede proporcionar directamente el fabricante del dispositivo, pero el usuario siempre tiene la opción de poder configurarlos como prefiera.

Los parámetros de los que debe disponer un dispositivo final tras ser activado y haberse unido a una red LoRaWAN correctamente son los siguientes:

- **Dirección del dispositivo (DevAddr)**: Dirección lógica dinámica (equivalente a una dirección IP) que se utiliza para toda comunicación con la red, esta dirección consta de 32 bits. En los siete bits más significativos figura el identificador de red (NwkID) utilizado para diferenciar una red del resto, los demás bits se corresponden con la dirección de red del nodo (NwkAddR), esta es asignada por la propia red [15].

Bit#	[31..25]	[24..0]
DevAddr bits	NwkID	NwkAddr

Figura 20. Formato DevAddR

- **Identificador de aplicación (AppEUI)**: Este parámetro se utiliza en el dispositivo final antes de que se ejecute el procedimiento de activación mediante el método OTAA, AppEUI identifica de forma única a la entidad encargada de procesar el mensaje JOIN [15].

- **Clave de sesión de red (NwkSKey)**: Clave de sesión de 128 bits única para cada nodo, es utilizada por el servidor de red y el dispositivo final para validar y garantizar la integridad de cada mensaje, esto se consigue comprobando el MIC (Código de integridad del mensaje) de todos los mensajes [5,16].

- **Clave de sesión de aplicación (AppSKey)**: Clave única de 128 bits para cada dispositivo final. Se utiliza para cifrar y descifrar el campo de carga útil de los mensajes entre el nodo final y la aplicación. Así se garantiza que ni la puerta de enlace ni el servidor de red puedan leer los datos enviados a y por el usuario.

A continuación, se muestra una imagen que refleja como es la comunicación de un nodo final una vez ya pertenece a una red LoRaWAN.

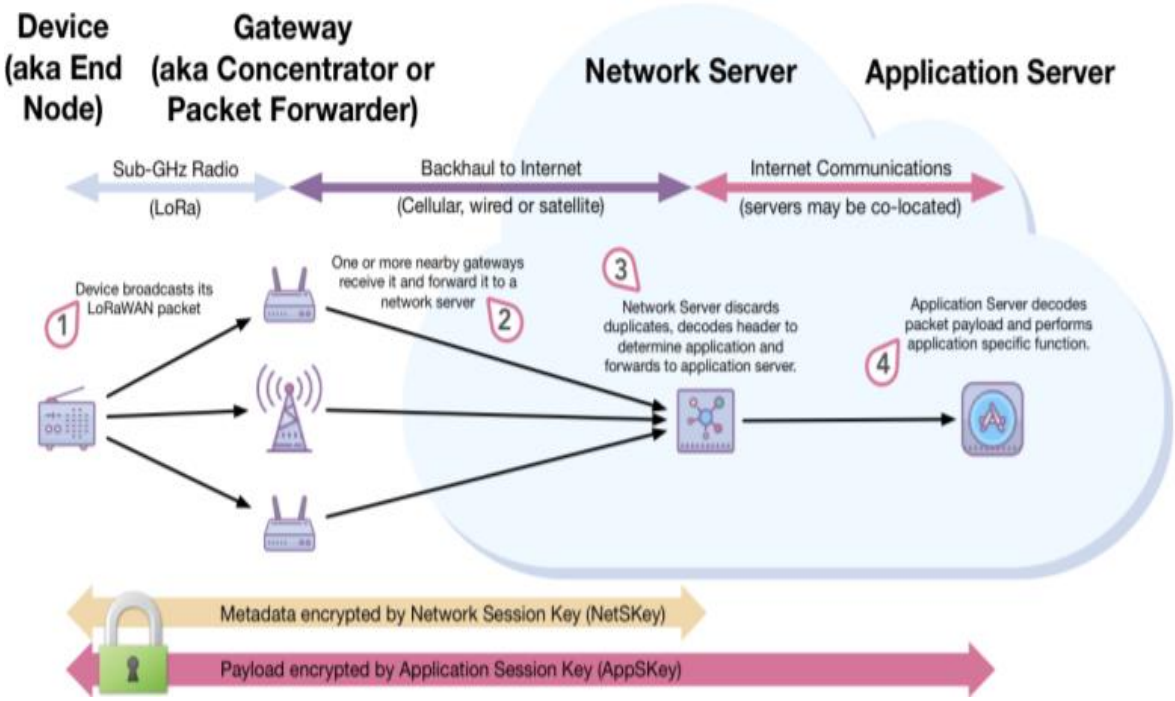

Figura 21. Comunicación en una red LoRaWAN con Chirpstack [17].

La activación de los dispositivos finales para que se puedan unir a una red LoRaWAN se puede realizar utilizando dos métodos diferentes:

5.1.1 ABP (Activation by Personalization)

Este método consiste en fijar previamente los parámetros DevAddr, NwkSKey y AppSKey, de forma personalizada por el usuario en el programa que se va a ejecutar en el dispositivo que se pretende unir a la red y en el propio servidor de la red. De esta forma el dispositivo final ya dispone de toda la información necesaria para participar en una red LoRaWAN [16].

El proceso que realiza este método para establecer la conexión es el siguiente:

1. El dispositivo envía sus datos (parámetros de conexión) al Gateway.
2. El Gateway valida que los datos enviados sean correctos y se puedan asignar a una sesión.
3. Si la sesión es correcta se establece la conexión, en caso contrario, se rechaza la conexión con ese dispositivo [16].

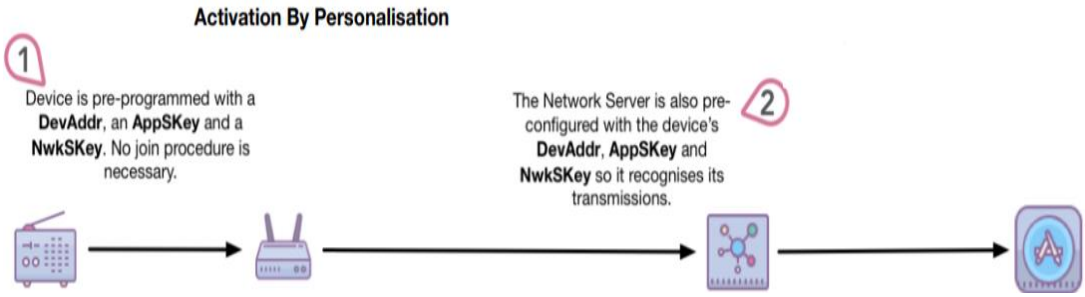

Figura 22. Activación por método ABP [17].

La principal ventaja de este tipo de conexión es que no se requiere hacer un *join* a la red para poder enviar datos, la confirmación del lado del servidor no es necesaria ya que los datos de la sesión ya están manualmente asignados; para dispositivos que no están destinados a la recepción de mensajes y dispositivos en constante movimiento este tipo de conexión es idónea [16].

Una desventaja aparece al encontrarse las llaves de encriptación en el dispositivo, estas pueden ser extraídas y clonadas por una atacante. Otra desventaja importante ocurre cuando aparecen eventos que requieren un cambio de claves (por ejemplo: mudarse a una nueva red, el dispositivo está comprometido o las claves caducadas), ante estas situaciones es necesario una reprogramación del dispositivo [16].

5.1.2 OTAA (Over-the-Air Activation)

El método OTAA (activación inalámbrica) es el método que se ha utilizado en este proyecto.

Este modo proporciona mayor seguridad que el modo ABP a la hora realizar una conexión a una red LoRaWAN. Para llevarlo a cabo, los dispositivos finales deben seguir un procedimiento de unión antes del intercambio de mensajes, el cual requiere la personalización del dispositivo final y la red con los siguientes parámetros:

- **DevEUI**

  Este parámetro es un identificador de 64 bits que hace único al nodo final, diferenciándolo del resto. Normalmente suele venir asignado de fábrica, aunque puede personalizarse [16].

- **AppKey**
  Es una clave secreta de 128 bits compartida entre el dispositivo final y la red. Esta clave es utilizada para a partir de ella generar las claves de sesión NwkSKey y AppSKey que se encargarán de asegurar la integridad y veracidad de los mensajes intercambiados entre el nodo final y la aplicación [16].

Una vez la red y el nodo final están correctamente configurados y se procede a conectar el dispositivo se activa el procedimiento de unión, en el cual se intercambian dos mensajes de tipo MAC: Join Request y Join Accept [17].

Los pasos que se siguen en el procedimiento de unión son:

1. El nodo solicita un join (unirse) a la red con los datos de configuración asignados (Join Request) y abre una ventana de recepción a la espera de un mensaje desde el servidor.
2. El Gateway recibe el mensaje del nodo final y se lo envía al servidor.
3. El servidor verifica que el nodo este dado de alta en el sistema y la clave de encriptación sea correcta.
4. Si la clave es correcta asigna una sesión temporal y la envía por medio del gateway al nodo (Join Accept), si los datos son incorrectos rechaza el join.
5. El nodo recibe la sesión temporal con su dirección dentro de la red (DevAddr) asignada y genera sus claves NwkSKey y AppSKey a partir de la respuesta del servidor, para poder transmitir en la red [16].

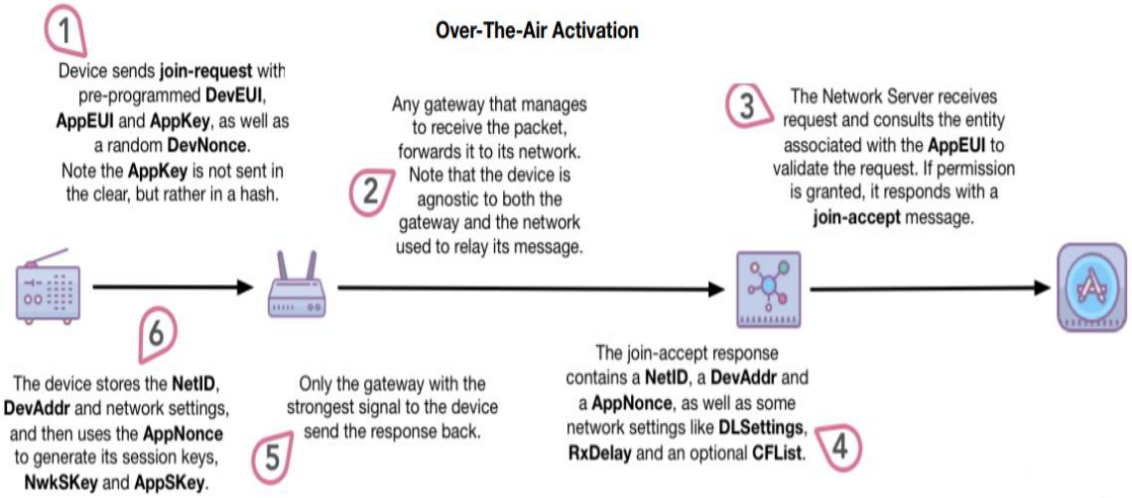

Figura 23. Activación por método OTTA [17].

La ventaja principal de este tipo de conexión es que la sesión es muy segura, ya que se crea bajo demanda y es renovada cada vez que el dispositivo se apaga, reinicia, o pierde la conexión. Por tanto, las claves de sesión solo se generan cuando son necesarias, por lo que no pueden ser accesibles antes del registro del dispositivo en la red [16].

5.1.3 Activación OTAA con cubecell AB02 en Chirpstack

Esta sección recoge un ejemplo del proceso que se sigue habitualmente para registrar un nodo cubecell AB02 en el sistema Chirpstack utilizando el método OTAA.
Los pasos que se siguen son los siguientes:

1. Creación de un nuevo dispositivo en Chirpstack

Lo primero es seleccionar una aplicación en la que crear un nuevo dispositivo, (sino existe ninguna aplicación en Chirpstack hay que crearla). Las aplicaciones se utilizan como agrupaciones de diferentes dispositivos que están siendo utilizados para un mismo proyecto. Chirpstack es capaz de gestionar varias aplicaciones.

Una vez dentro de la aplicación, hay que seleccionar “Create” y completar los campos correspondientes en el formulario, asignando un nombre al dispositivo, una breve descripción y un Device EUI (debe ser igual al que se asignará en el nodo final cubecelll AB02), también es necesario utilizar un device profile, este define las capacidades del dispositivo y los parámetros de arranque que necesita el servidor de red para configurar el servicio de acceso de radio LoRaWAN.

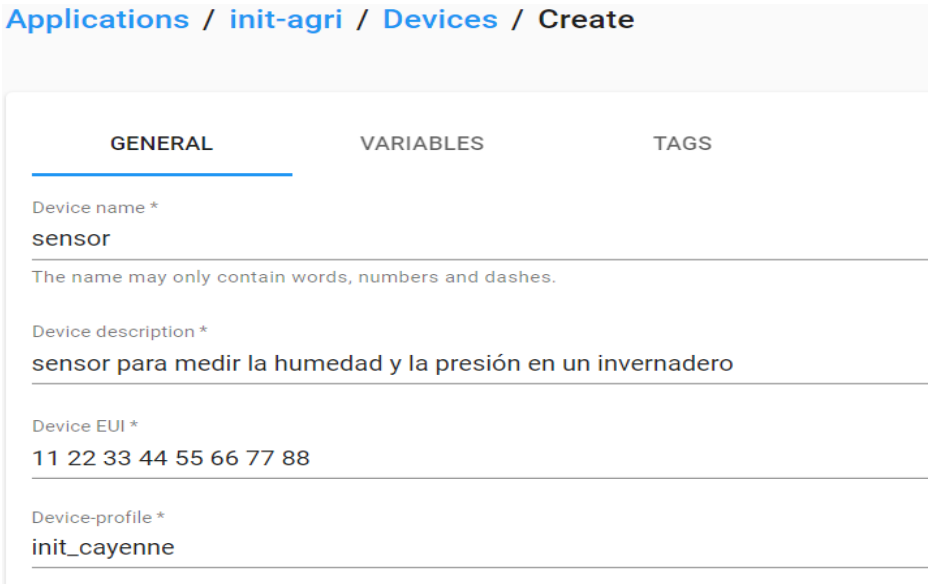

Figura 24. Ejemplo de creación de un dispositivo en Chirpstack.

Una vez ya está creado el dispositivo en Chirpstack hay asignarle un application key (debe ser igual al appKey asignado en el nodo final cubecell AB02).

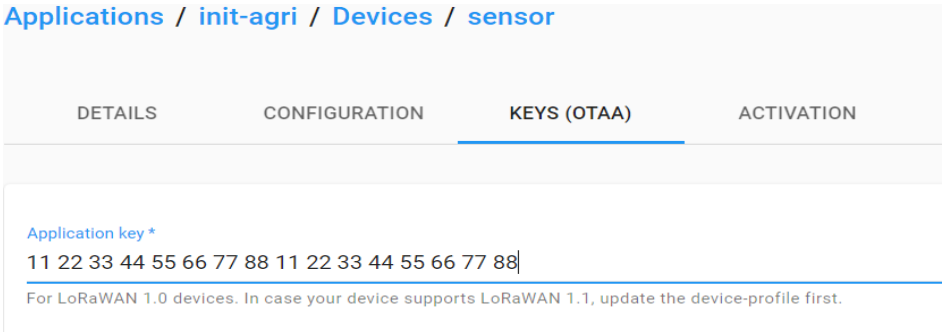

Figura 25. Ejemplo de asignación AppKey a un dispositivo en Chirpstack

Finalmente, ya se dispone del dispositivo registrado en Chirpstack, aunque esto no indica que un nodo final ya este capacitado para comunicarse con la red, ya que antes es necesario configurar este.

2. Configuración de un nodo cubecell AB02

Conociendo que el módulo cubecell es compatible con la tecnología Arduino, se utilizara esta para desarrollar el programa que será necesario que la placa ejecute para su correcto funcionamiento. Este programa debe incluir todas las funciones requeridas al dispositivo, por ejemplo: lectura de un sensor asociado, intervalo de tiempo entre transmisiones, parameros para unirse a una red LoRaWAN, etc…

Los parámetros de ejemplo del nodo para poder unirse y ser asociado con el dispositivo creado en el apartado anterior en Chirpstack se muestra en la siguiente imagen:

```
#include "LoRaWan_APP.h"
#include "Arduino.h"

/* OTAA params*/
uint8_t devEui[] = { 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88};
uint8_t appEui[] = { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 };
uint8_t appKey[] = {0x11,0x22,0x33,0x44,0x55,0x66,0x77,0x88,0x11,0x22,0x33,0x44,0x55,0x66,0x77,0x88};

bool overTheAirActivation = LORAWAN_NETMODE;
```

Figura 26. Trozo de código de configuración de parámetros LoRaWAN en un nodo final.

Una vez ya se dispone del nodo final correctamente configurado, este puede ponerse en marcha y se registrara correctamente en la red LoRaWAN, siendo asociado al dispositivo creado en Chirpstack.

Como se ha visto, la configuración manual de un dispositivo final teniendo en cuenta que se va a realizar un despliegue masivo de estos, es una tarea tediosa de realizar que requiere una gran cantidad de tiempo, ya que es necesario instalar el mismo firmware en cada dispositivo, siendo necesario asignar a cada uno de ellos un devEUI y un appKey diferentes del resto de dispositivos.

5.2. Diseño de la arquitectura software

Ahora que ya se conoce con detalle todo el procedimiento que se realiza para que un dispositivo final pueda ser registrado en Chirpstack, se desarrolla el diseño de una arquitectura software que pueda agilizar y automatizar este proceso. La arquitectura software diseñada debe permitir que todo el proceso del registro de un dispositivo LoRaWAN (cubecell AB02) en Chirpstack sea mucho más rápido y eficaz, además se pretende que se identifique la ubicación de cada dispositivo cuando es registrado, de esta forma se puede saber que los datos recogidos y enviados por un posible sensor corresponden a una localización determinada.

Contando con las características de esta arquitectura, un despliegue masivo de dispositivos LoRaWAN (cientos o miles) sería mucho más asequible, ya que su registro sería una tarea menos tediosa y se conocería la ubicación desde la que cada dispositivo trasmite los datos.

La arquitectura software se diseña entorno a cuatro elementos diferentes que deben comunicarse entre ellos para poder establecer toda la infraestructura necesaria para conseguir los objetivos planteados. Los elementos que componen esta arquitectura son los siguientes:

- **Nodo final LoRa**

  Es el dispositivo cubecell AB02, la idea es que se pueda realizar una compra al fabricante en la que todos los dispositivos adquiridos contengan el mismo firmware. El firmware que deben contener todos los dispositivos es el desarrollado en este proyecto.

  El desarrollo del firmware para el módulo cubecell AB02 consiste en aprovechar que el dispositivo dispone de una pantalla OLED, la idea principal es utilizar esta pantalla para que en ella se pueda mostrar el DevEUI y el AppKey del dispositivo mediante un código QR.

  Todos los módulos cubecell disponen de una instrucción que permite obtener un DevEUI para el dispositivo según el chip que ha instalado el fabricante en este, por tanto, se usará esta instrucción para obtener un DevEUI que identifique a cada dispositivo y lo haga diferente del resto. Este DevEUI sería el equivalente a una dirección MAC ya que debe ser único para cada dispositivo. En cuanto al AppKey no se dispone de ninguna instrucción similar a la anterior, por tanto, se ha diseñado un método que permita generar un AppKey aleatorio para cada dispositivo. Una vez se dispone del AppKey este será almacenado en la memoria EEPROM del dispositivo, de esta forma si el dispositivo se resetea no se generará un nuevo AppKey, sino que este se leerá de la memoria y será siempre el mismo.

Una vez el dispositivo dispone del AppKey y el DevEUI mostrara en su pantalla OLED ambos parámetros hasta que consiga conectarse a la red LoRaWAN, cuando finalmente el dispositivo consigue conectarse se mostrará un mensaje de confirmación y se apagara la pantalla para ahorrar batería.

La forma que los dispositivos utilizaran para mostrar el AppKey y el DevEUI en la pantalla OLED será mediante un código QR. Este código QR representara de forma codificada ambos parámetros, de esta manera se puede conocer cuáles son los valores asignados al DevEUI y al AppKey del dispositivo simplemente escaneando el código QR.

- **App movil (ChirpRegister)**
  Este elemento es una aplicación móvil Android. Esta aplicación debe permitir a los usuarios escanear el código QR mostrado en la pantalla OLED de un nodo final, la correcta decodificación del QR permite obtener tanto el DevEUI como el AppKey asociado al dispositivo. Una vez se dispone del DevEUI y el AppKey del dispositivo en cuestión, el usuario puede obtener las coordenadas de su posición actual utilizando el GPS de su teléfono móvil. Posteriormente toda esta información se enviará a la App web Chirpcontrol, esta recibirá los datos y enviara una respuesta indicando si la acción ha tenido éxito y se ha podido registrar el dispositivo, o en cambio, ha surgido algún error en el proceso, como por ejemplo algún campo incorrecto o que ya existe un dispositivo registrado con ese DevEUI.

  La app móvil, aparte de lo comentado anteriormente, permitirá a los usuarios gestionar todos los dispositivos que hay registrados en ese momento, pudiendo tanto editarlos como eliminarlos. También permitirá consultar un mapa en el que se pueda ver la ubicación de cada dispositivo registrado.

- **App web (Chirpcontrol)**
  Chirpcontrol es el nombre de la aplicación web desarrollada en este proyecto. Su función principal es actuar como intermediario entre la aplicación móvil (ChirpRegister) y el sistema de gestión/explotación de los dispositivos finales, en este caso Chirpstack.

  Todas las peticiones que realiza un usuario a través de la aplicación móvil ChirpRegister son enviadas a Chirpcontrol, este se encarga de validar si las peticiones siguen el formato correcto y comunicarse finalmente con la API de Chirpstack para terminar de realizar la solicitud enviada por el usuario.

Además, Chirpcontrol dispone de una vista web con un mapa en el que se encontraran ubicados todos los dispositivos registrados en Chirpstack a través de la app móvil, siempre y cuando el usuario haya optado por añadir las coordenadas de su ubicación cuando realiza el registro del dispositivo. Esto soluciona la limitación de Chirpstack en la que únicamente se pueden ubicar en un mapa los gateways que forman parte de la red, pero no los nodos de campo finales.

En la parte final del proyecto a Chirpcontrol se le añadió una interfaz web de usuario, el objetivo de esta interfaz es proporcionar la misma función que la aplicación móvil, pensado para aquellos usuarios que no dispongan de los requerimientos para instalar la aplicación móvil ChirpRegister, el caso de gente que no disponga de teléfonos Android o que por algún motivo no desee instalarse una nueva aplicación en su dispositivo.

- **Chirpstack API**
  Es un componente integrado en Chirpstack. Su objetivo principal es permitir a terceros el acceso y la gestión de todos los dispositivos registrados Chirpstack. Por tanto, Chirpstack API es una herramienta que facilita enormemente la comunicación de Chirpstack con otras aplicaciones o sistemas externos.

  Este elemento de la arquitectura software propuesta, a diferencia de los otros tres, no hay que desarrollarlo mediante código, la labor es estudiar todos los métodos que proporciona Chirpstack API e identificar cuáles son los que se van a necesitar utilizar en cada momento para el desarrollo del proyecto. Siendo también necesario el conocimiento sobre cuáles son los requerimientos y la metodología a seguir para poder consumir de forma adecuada Chirpstack API.

Una vez ya han sido definidos y explicados brevemente los elementos que componen la arquitectura software desarrollada en este proyecto, se ha diseñado un diagrama de actividades que demuestra de una forma visual como los elementos se relacionan y comunican entre ellos ante una posible situación en la que se realiza la unión de un nodo final LoRa a una red LoRaWAN gestionada por Chirpstack mediante la arquitectura desarrollada en este proyecto.

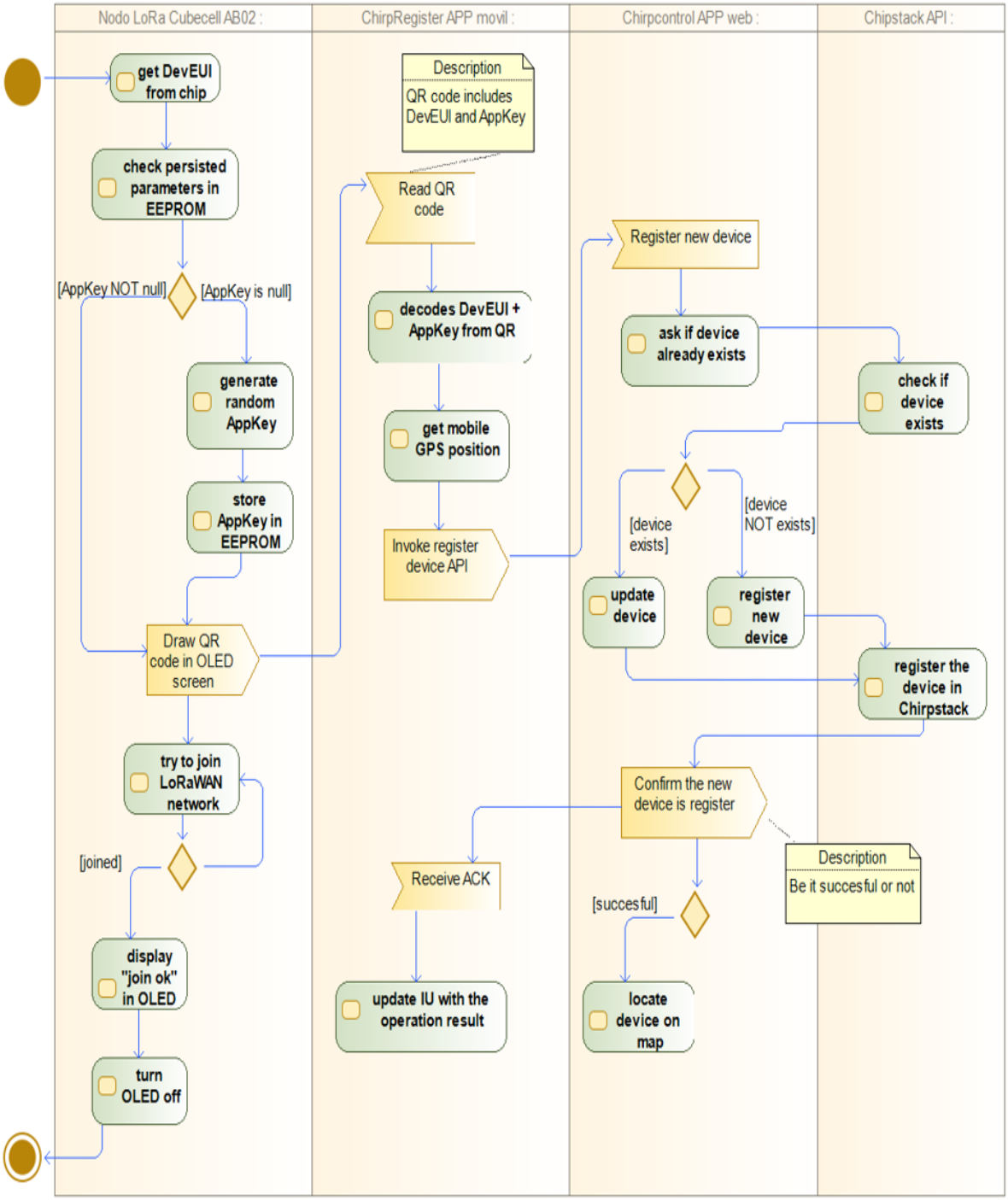

Figura 27. Diagrama de actividades, JOIN procedure.

5.3. Desarrollo de Chirpcontrol (APP web externa).

Chipcontrol es una aplicación web desplegada en un servidor Ubuntu 20.04 que tiene como función actuar de intermediario entre la app móvil "ChirpRegister" y Chirpstack API. Debido a esto su desarrollo básicamente se divide en estas secciones:

5.3.1 Comunicación con Chirpstack API

Como ya se ha comentado, Chirpstack incluye una API REST (Chirpstack API) que permite su comunicación con aplicaciones externas. Chirpstack API proporciona una interfaz web que contiene todos sus métodos API y su documentación asociada. El acceso a esta interfaz web se puede hacer añadiendo "/api" en el navegador a la dirección URL en la que se encuentra ubicado Chirpstack. Por ejemplo: *http://localhost:8080/api*.

En esta interfaz web se pueden consultar todos los métodos API que dispone Chirpstack, estos métodos se encuentran agrupados por secciones según el elemento de Chirpstack con el que interactúan (aplicaciones, dispositivos, usuarios, gateways, etc…).

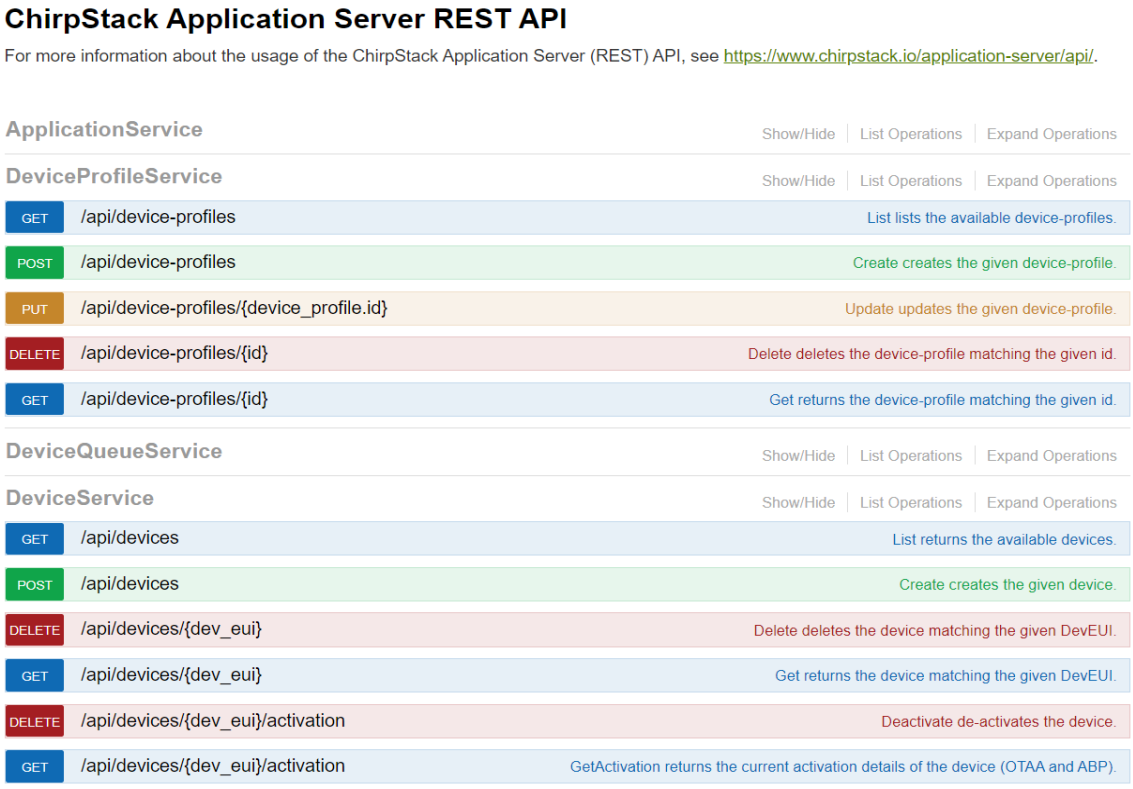

Figura 28. Interfaz web de Chirpstack API que muestra todos sus métodos.

Cada método API dispone de una URL, produce un resultado y tiene unos requerimientos o parámetros (información que necesita para poder ser ejecutado correctamente). Los detalles de cada método API se pueden consultar en esta interfaz web. La siguiente imagen es un ejemplo en el que se muestra en detalle un método API, este se utiliza para obtener los datos de un dispositivo en concreto. Para que este método funcione de forma correcta es necesario indicar el devEUI del dispositivo a consultar como parámetro. Como

resultado esperado de la solicitud se devuelve un JSON que contiene toda la información del dispositivo.

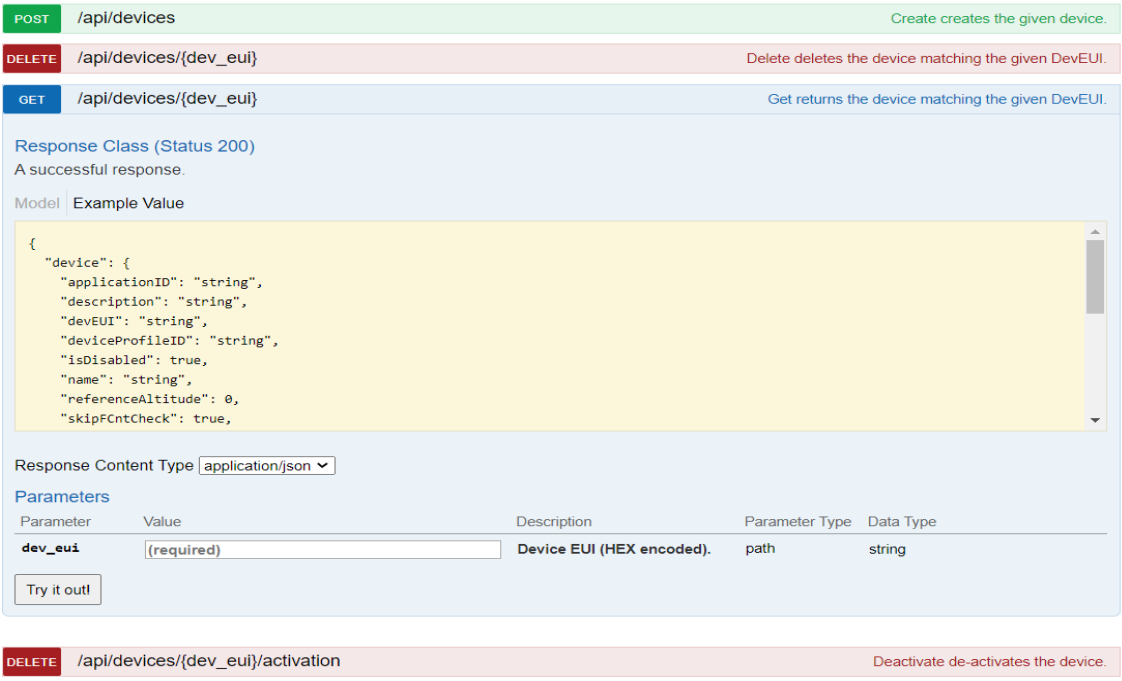

Figura 29. Consulta de un método API en la interfaz web de Chirpstack API.

Una vez ejecutado el método API habiéndole asignado como parámetro dev_eui el valor 1111111111111111 se obtiene el siguiente resultado:

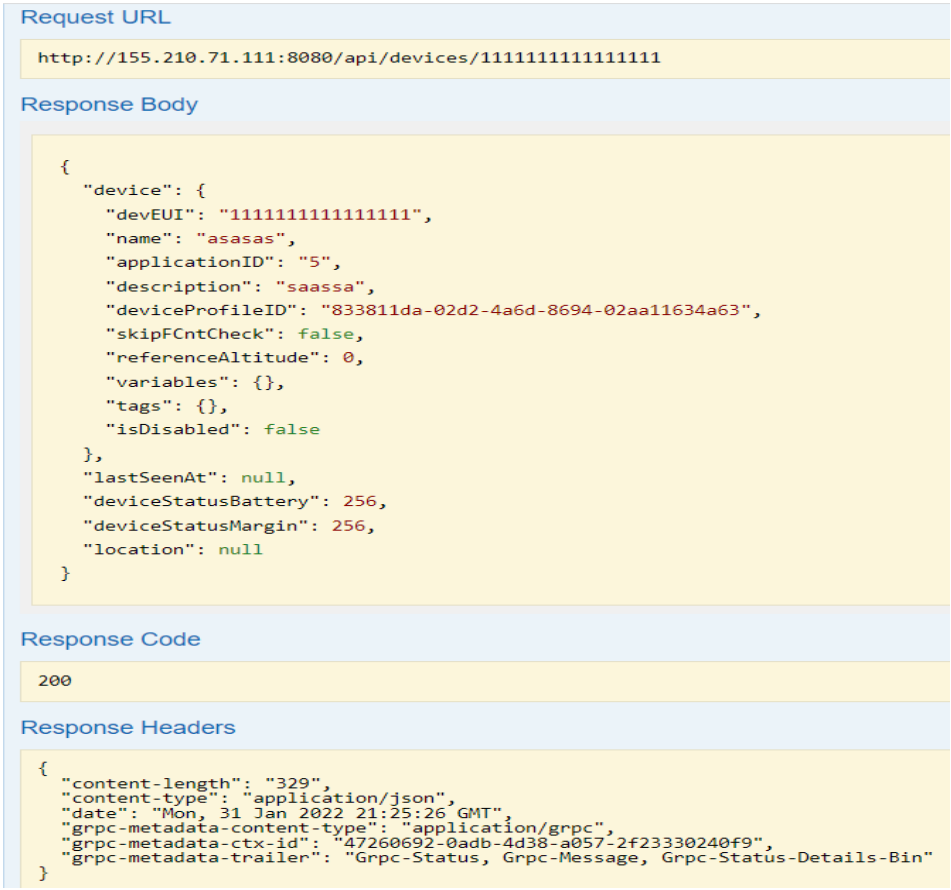

Figura 30. Ejecución de un método API en la interfaz web de Chirpstack API.

Se puede observar que el método API devuelve un código de respuesta 200 que indica que el método se ha ejecutado con éxito y un JSON con toda la información relativa al dispositivo consultado.

Hay que destacar que el uso de los métodos API de Chirpstack está restringido, para poder ser utilizados es necesario disponer de un token. Este token debe ser introducido en la casilla "JWT TOKEN" de la interfaz web antes de ejecutar cualquier método API, en caso contrario, al ejecutar un método aparecerá un error de autenticación.

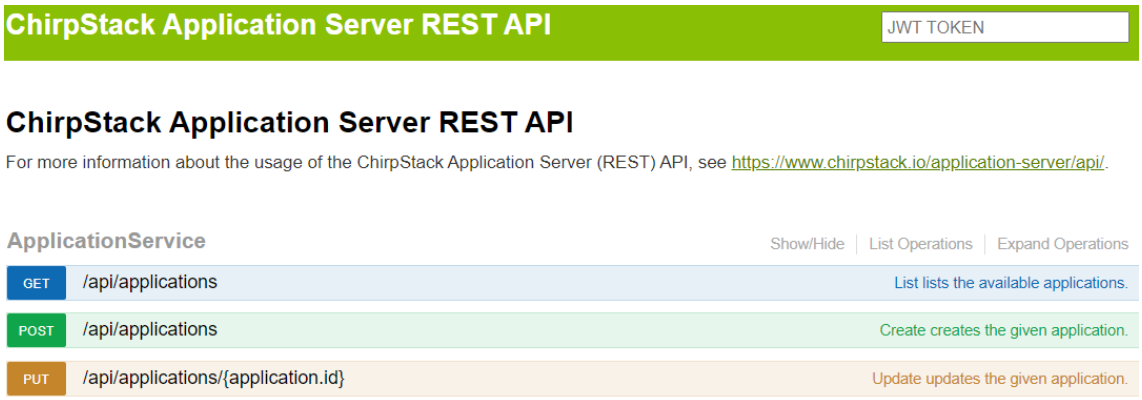

Figura 31. Casilla JWT TOKEN en la interfaz web de Chirpstack API.

Para disponer de un token es necesario iniciar sesión en Chirpstack, entrar en la pestaña "API keys" y elegir un API key ya existente, sino existe, sería necesario crear una nuevo.

El siguiente paso, tras ya haberse realizado una primera toma de contacto con la interfaz web de Chirpstack API, consiste en seleccionar cuales van a ser los métodos API cuyo uso va a ser fundamental en la realización de este proyecto. Chirpstack API dispone de una gran cantidad de métodos, sin embargo, en este proyecto solo van a ser necesarios algunos de ellos, siendo la mayoría de estos pertenecientes a la sección de dispositivos.
Finalmente, los métodos API que se han sido necesarios en este proyecto son los siguientes:

- Métodos GET

Estos métodos son los utilizados para obtener información de Chirpstack y en la mayoría de los casos retornan la información en formato JSON.

Los métodos get empleados en este proyecto se pueden dividir en dos apartados, los destinados a consultar información sobre el conjunto de elementos que hay registrados en una sección. Por ejemplo: *obtener información de todos los dispositivos registrados en Chirpstack*; y los destinados a consultar información sobre un elemento en específico. Por ejemplo: *obtener información de un dispositivo en concreto*. Los primeros retornan un vector de elementos. Pero no devuelven todos los elementos existentes, sino que disponen de un parámetro llamado "limit" que indica el número máximo de elementos que debe contener el vector que se va a retornar. Por tanto, si se quiere retornar un vector que incluya todos los elementos se debe indicar el número total de elementos o un número

superior a este en el parámetro “limit”. Si este parámetro es nulo o se le asigna un valor igual a 0 el vector a retornar estará vacío.

La respuesta que producen estos métodos es un JSON con estas dos propiedades:

- result: Vector formado por todos los elementos devueltos y su información asociada.
- totalCount: Número de elementos que existen en total.

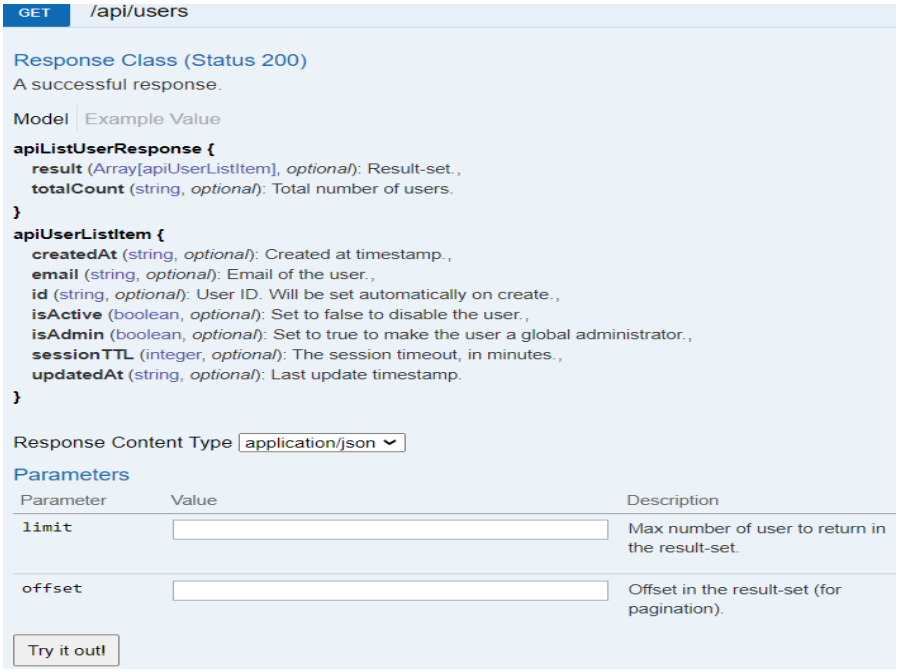

Figura 32. Devolución de información todos los usuarios registrados en Chirpstack.

Los segundos devuelven un JSON que almacena toda la información relativa a un elemento en concreto. Como parámetro normalmente solo necesitan un identificador, para poder encontrar el elemento a consultar entre el resto.

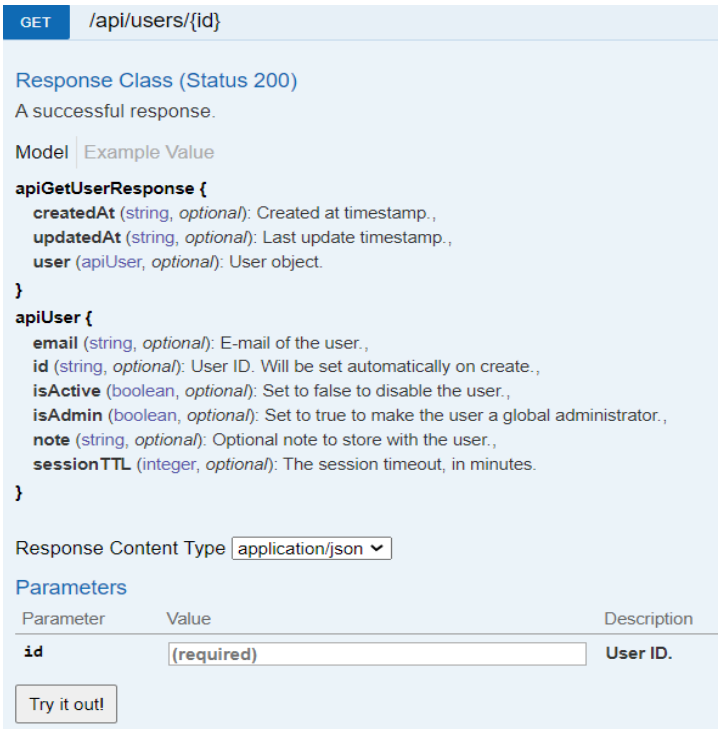

Figura 33. Devolución de información de un usuario especifico.

Finalmente, los métodos GET utilizados en este proyecto son los siguientes:

- **Get** api/applications

  Devuelve todas las aplicaciones que hay creadas en Chirpstack. La siguiente imagen refleja la estructura de este método API:

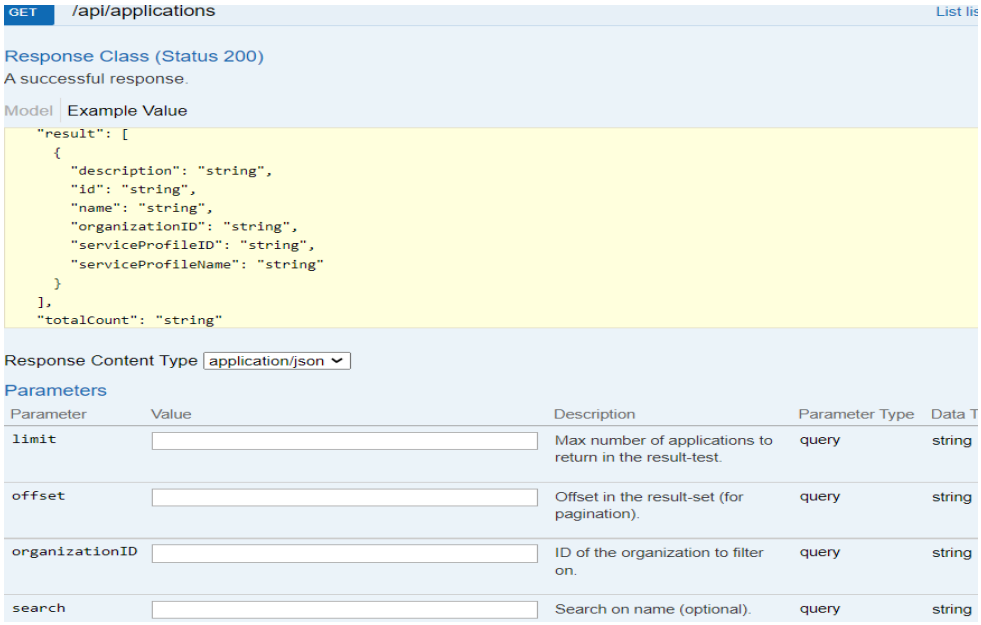

Figura 34. Estructura método API get applications.

El único parámetro que se utilizado en este método API es "limit", el valor asignado a este ha sido 25 para asegurar que se devuelvan todas las aplicaciones, ya que es poco probable que puedan existir más de diez aplicaciones.

- **Get** api/device-profiles

  Este método devuelve todos los devices profiles existentes en Chirpstack. El funcionamiento de este método es igual al anterior (Get api/applications) ya que también se le ha asignado el valor 25 a limit y solo se utiliza este parámetro.

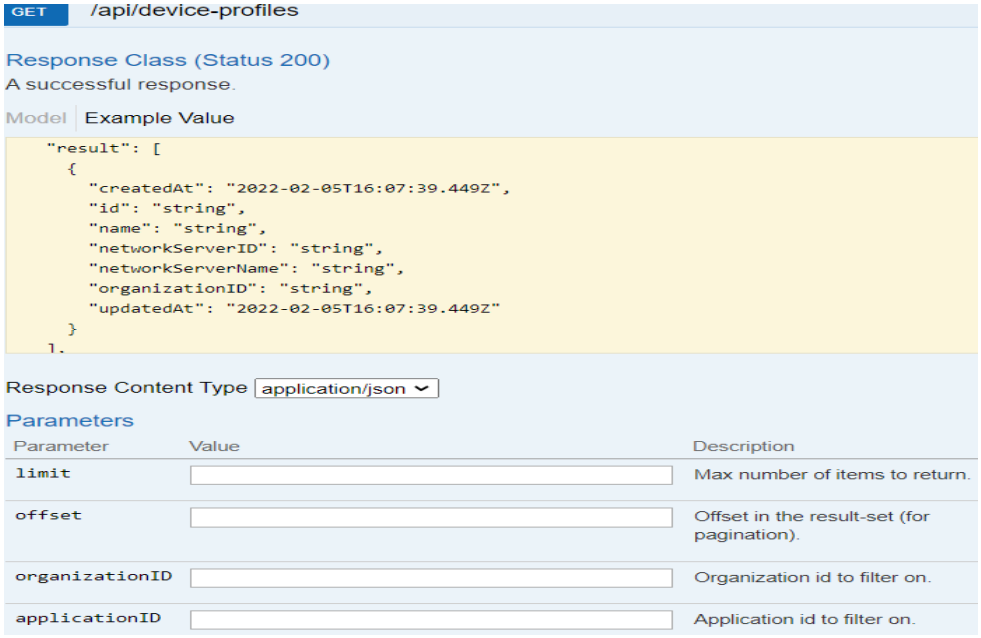

Figura 35. Estructura método API get device-profiles.

- **Get** api/devices

  Los dispositivos en Chirpstack se encuentran agrupados por aplicaciones. Este método devuelve la información de los dispositivos que se encuentran registrados en una aplicación concreta de Chirpstack.

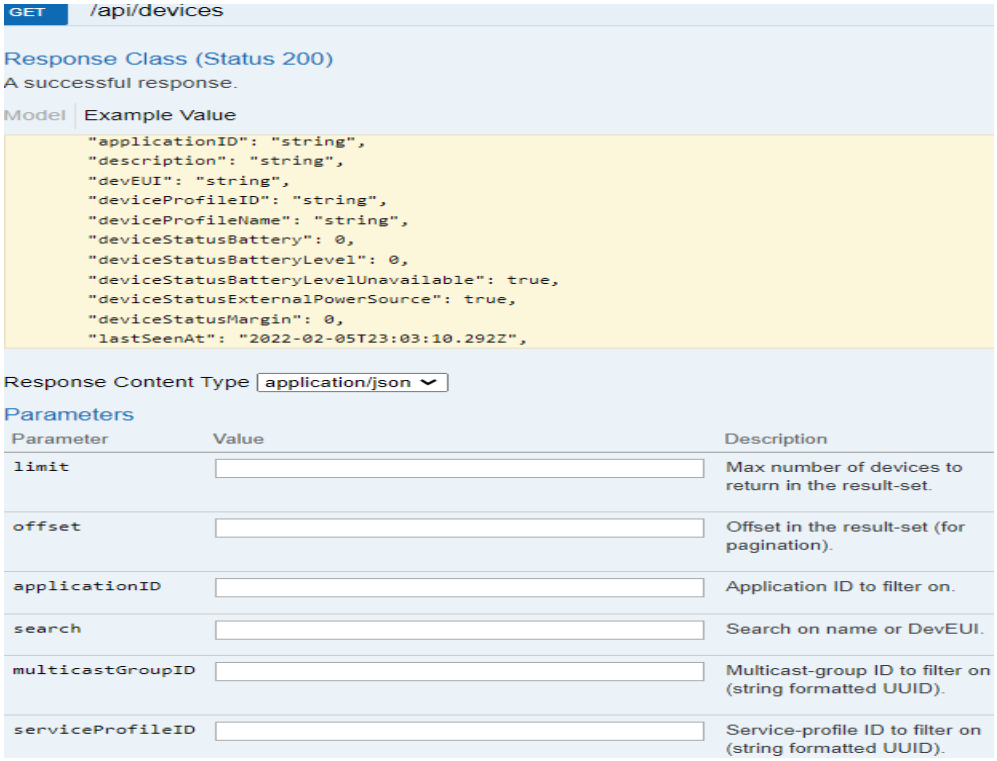

Figura 36. Estructura método API get Devices.

En este método API se han utilizado los parámetros limit y applicationID. El parámetro applicationID es siempre el mismo ya que en este proyecto solo se trabaja con una aplicación. En cuanto al parámetro limit, a diferencia de los métodos anteriores, no se le ha asignado el valor 25 por defecto, sino que se asigna de esta forma; primero se hace una llamada al propio método *get /api/devices* con el valor limit igual a cero, de esta forma se obtiene el JSON de respuesta con las propiedades result y totalCount, result estará vacío y totalCount devuelve el número de dispositivos que están registrados. Una vez se dispone del valor de totalCount se hace una segunda llamada al método *get /api/devices* asignando al parámetro limit el valor de totalCount y al parámetro applicationID el identificador de la aplicación a consultar (mismo valor que en la llamada anterior). Obteniendo así en la propiedad result del JSON devuelto la lista de todos los dispositivos que hay registrados en la aplicación de Chirpstack consultada.

La forma de ejecutar este proceso es menos eficiente que las anteriores ya que se necesitan realizar dos llamadas al método API, sin embargo, se asegura que el valor asignado a limit siempre sea el correcto, evitando así cualquier riesgo que pueda suponer que la lista devuelta de todos los dispositivos no este completa, ya el número total de dispositivos registrados va a ser muy cambiante.

- **Get** /api/devices/{dev_eui}

  Este método devuelve la información asociada a un dispositivo concreto registrado en Chirpstack. Por tanto, para que funcione correctamente es necesario indicar el DevEUI del dispositivo a consultar como parámetro, de esta forma se encuentra este dispositivo concreto entre todos los que están registrados.

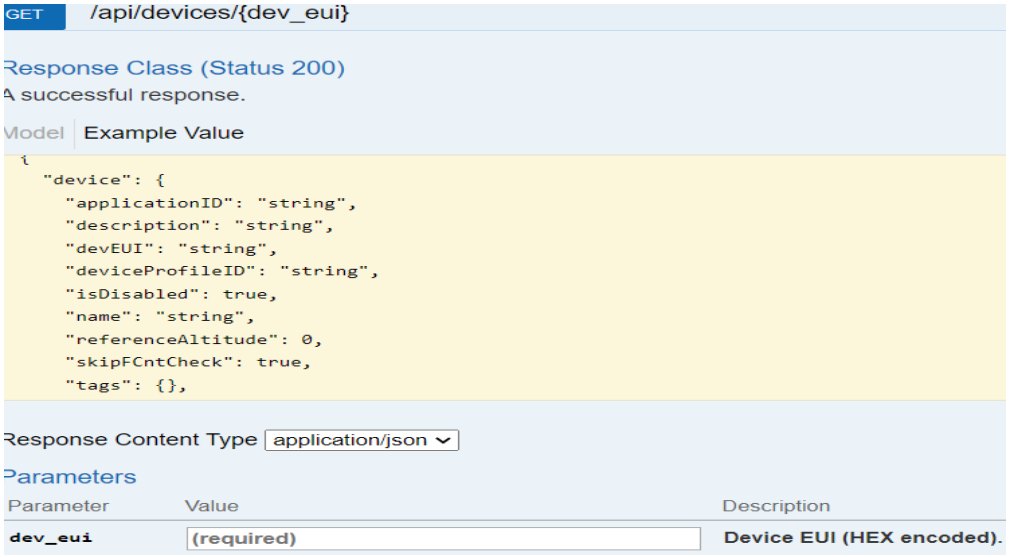

Figura 37. Estructura del método API get/device/{dev_eui}.

- **Get** /api/devices/{dev_eui}/keys

  Devuelve las "keys" asociadas a un dispositivo concreto. Este método es utilizado para conseguir el appKey asignado a un dispositivo. Como parámetro recibe el DevEUI del dispositivo a consultar.

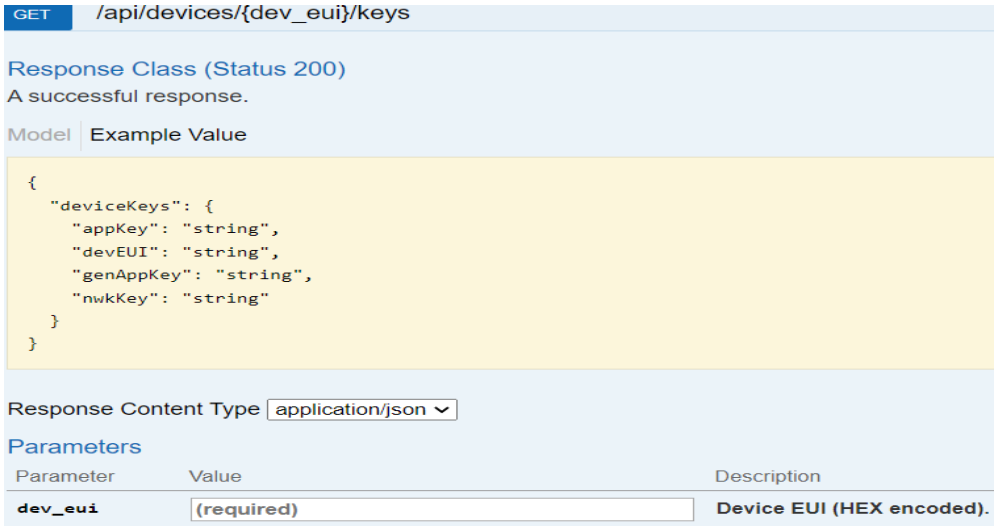

Figura 38. Estructura del método API Get /api/devices/{dev_eui}/keys.

Todo hace indicar que la clave appKey buscada se encuentra en el campo "appKey" del JSON devuelto, sin embargo, es en el campo "nwkKey" donde esta. Esto ocasiono algún problema al principio, pero finalmente consultando documentación se encontró información que indica que el campo appKey es solo

para dispositivos de versión 1.1.x y que nwkKey es utilizado para generar las claves de sesión nwkSKey y appSkey, es decir, se corresponde con el termino appKey utilizado hasta el momento.

- Metodos POST

Utilizados para añadir información a Chirpstack. Los métodos API utilizados son:

- **Post** /api/devices

  Se utiliza para registrar un nuevo dispositivo en Chirpstack. Como parámetro necesita un JSON que contenga toda la información del nuevo dispositivo.

Figura 39. Estructura del método API POST /api/devices.

Del modelo de ejemplo JSON mostrado en la imagen solo va a ser necesario completar los campos applicationID, description, devEUI, deviceProfileID y name para poder registrar un nuevo dispositivo.

- **Post** /api/devices/{device_keys.dev_eui}/keys

  Este método es utilizado para asignar el appKey a un dispositivo ya registrado en Chirpstack. Como parámetros necesita el DevEUI del dispositivo al que se le va a asignar el appKey y un JSON que contiene este appKey.

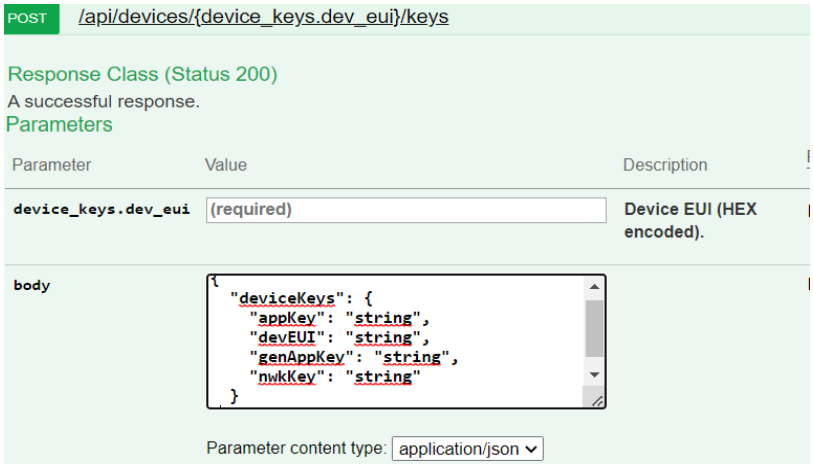

Figura 40. Estructura del método API Post /api/devices/{device_keys.dev_eui}/keys.

Del JSON a enviar solo es necesario asignar el campo nwkKey, que como ya se ha comentado anteriormente se corresponde con el appKey del dispositivo.

- Metodos PUT

Utilizados para actualizar información de Chirpstack. Los métodos utilizados son:

- **Put** /api/devices/{device.dev_eui}

  Actualiza la información de un dispositivo concreto, como parámetros necesita el DevEUI del dispositivo y un JSON que contiene los datos que se van a actualizar.

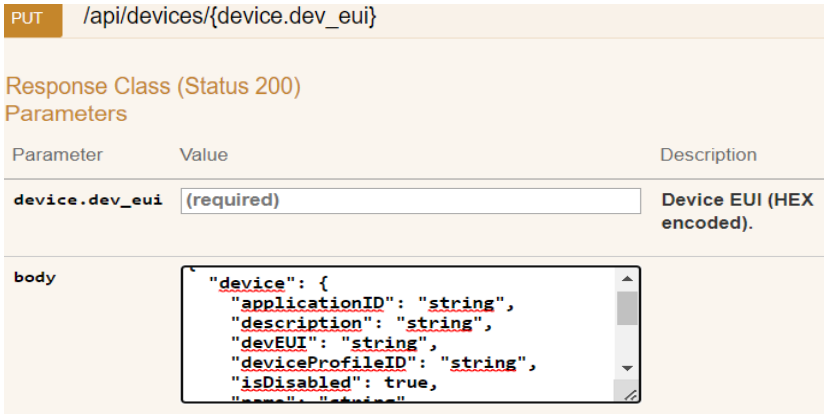

Figura 41. Estructura del método API Put /api/devices/{device.dev_eui}.

- **Put** /api/devices/{device_keys.dev_eui}/keys

  Actualiza el appKey de un dispositivo determinado, como parámetros necesita el DevEUI del dispositivo y un JSON que contenga el nuevo valor del appKey en el campo nwkKey del JSON, el resto de los campos pueden ser nulos sin problemas.

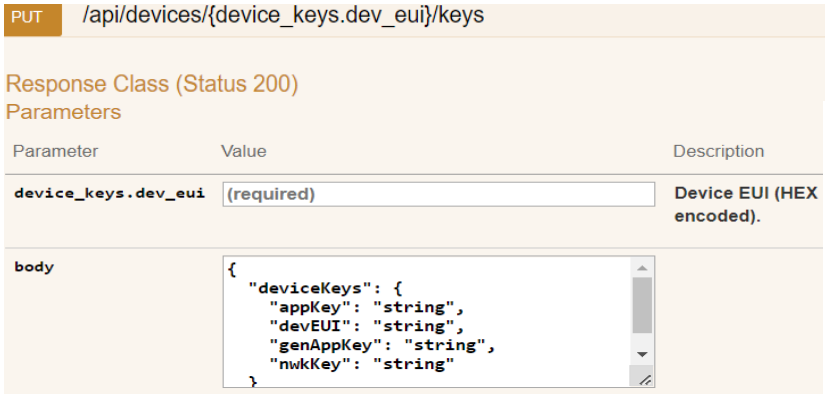

Figura 42. Estructura del método API Put /api/devices/{device.dev_eui}/keys.

- Metodos DELETE

- **Delete** /api/devices/{dev_eui}

  Utilizado para eliminar un dispositivo de Chirpstack, como parámetro necesita el DevEUI del dispositivo a eliminar.

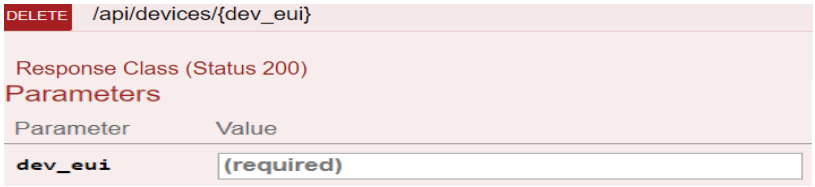

Figura 43. Estructura del método API Delete /api/devices/{dev_eui}.

Una vez analizados los métodos API que se requieren para realizar este proyecto, se realiza la llamada a cada uno de estos métodos desde código.

Recordando que las llamadas a estos métodos API están restringidas, para poder utilizar estos desde código se debe insertar en los encabezados de las peticiones la tupla *'Grpc-Metadata-Authorization' - 'JWT TOKEN'*, donde el valor de JWT token se leerá del fichero de configuración de Chirpcontrol. Esto es equivalente a rellenar la casilla 'JWT TOKEN' de la interfaz web, lo que permitía el acceso a realizar llamadas a todos los métodos API disponibles en Chirpstack.

A continuación, se muestran dos ejemplos de llamadas a métodos de Chirpstack API desde código.

```
public String getAppKey(String devEUI) {
    String urlKey = devicesURL+KEYS;
    String urlGetAppKey = String.format(urlKey,devEUI);

    webTarget = client.target(urlGetAppKey);
    Invocation.Builder solicitud = webTarget.request().header(header, token);
    Response get = solicitud.get();
    int status = get.getStatus();
    if(status == 200){
        String result = get.readEntity(String.class);
        JsonObject jo = new JsonParser().parse(result).getAsJsonObject();
        JsonObject deviceKeys = jo.get("deviceKeys").getAsJsonObject();
        String appKey = deviceKeys.get("nwkKey").getAsString();
        return appKey;
    }
    return null;
}
```

Figura 44. Llamada a método API de tipo GET para obtener el appKey de un dispositivo, sin tratar posibles errores en los punteros para reducir el contenido de la imagen.

```
public String addDevice(Device device) {
    String postJson = "{\"device\":"+device.toJson()+"}";
    webTarget = client.target(devicesURL);
    Invocation.Builder solicitud = webTarget.request().header(header, token);
    Response post = solicitud.post(Entity.json(postJson));
    int status = post.getStatus();
    if(status == 200){
        return device.getDevEUI();
    }
    return null;
}
```

Figura 45. Llamada a método API de tipo POST para registrar un nuevo dispositivo en Chirpstack.

5.3.2 Desarrollo de una API REST

Chirpcontrol debe proporcionar una API REST que pueda ser utilizada por la aplicación móvil "ChirpRegister", permitiendo así la comunicación entre ambos sistemas. El objetivo de esta API REST es permitir que cualquier sistema pueda comunicarse con Chirpcontrol de una forma sencilla, de esta forma si en algún otro momento se sustituyese la aplicación móvil ChirpRegister por otra aplicación o fuese necesario añadir algún otro elemento a la arquitectura software no habría ningún problema en comunicar los nuevos componentes con Chirpcontrol.

Como conclusión, Chirpcontrol proporciona esta API REST que unido a la capacidad que dispone para poder consumir métodos de Chirpstack API hace que actúe como el intermediario perfecto para permitir que sistemas de terceros puedan comunicarse con Chirpstack de forma directa, es decir, sin tener que preocuparse de disponer del token de autenticación.

5.3.3 Mapa de ubicaciones de los dispositivos.

Chirpcontrol proporciona un mapa en el que se encontraran ubicados todos los dispositivos registrados en Chirpstack a través de la app móvil, siempre y cuando el usuario haya optado por añadir las coordenadas de su ubicación cuando realiza el registro del dispositivo.

Para el desarrollo de este mapa se ha utilizado Google Maps API, esta es una librería JavaScript que permite insertar Google Maps en nuestras páginas web. Antes era un servicio gratuito, pero ahora ha pasado a ser de pago, aun así permite su uso para ciertos proyectos pero como consecuencia proporciona una visualización desmejorada de sus mapas. Aún con este pequeño inconveniente la librería se adapta sin problemas a las necesidades de este proyecto. Para poder utilizar tanto esta librería como todas las aplicaciones de la API de Google Maps se requiere de una clave. Por lo tanto, lo primero que fue necesario hacer es acudir a la página de Google y obtener una clave de API.

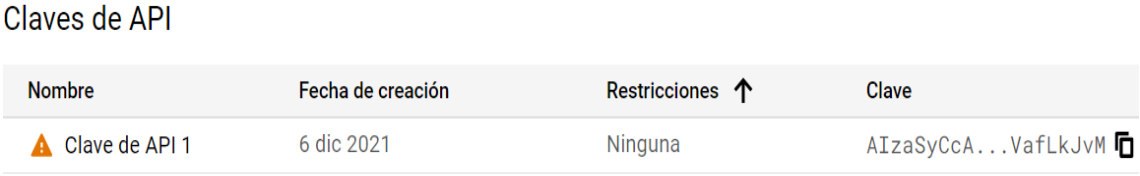

Figura 46. Obtención de una clave API de Google Maps.

Una vez ya se dispone de la clave API el mapa se ha desarrollado con las siguientes características. El centro del mapa se ha ubicado en la ciudad de Madrid y se ha utilizado un valor de zoom desde el que se pueda visualizar toda España. La idea es que el usuario parta de una visión general del mapa de España y pueda ir variando los valores de zoom

y centrándose en aquellos lugares en los que se haya realizado el despliegue masivo de los dispositivos LoRaWAN.

Cuando un dispositivo es registrado en Chirpstack, si el usuario ha decidido mandar las coordenadas de su ubicación, en el mapa se añadirá un marcador en las coordenadas enviadas por el usuario, este marcador mostrará el nombre del dispositivo para poder así relacionar cada dispositivo con su localización.

El mapa se ha desarrollado utilizando la tecnología WebSocket lo que permite la actualización en tiempo real de las ubicaciones de los dispositivos. De esta forma irán apareciendo y desapareciendo marcadores del mapa según se van creando, eliminando y modificando dispositivos. La siguiente imagen es un ejemplo que enseña cómo se vería en el mapa la localización de algunos dispositivos registrados.

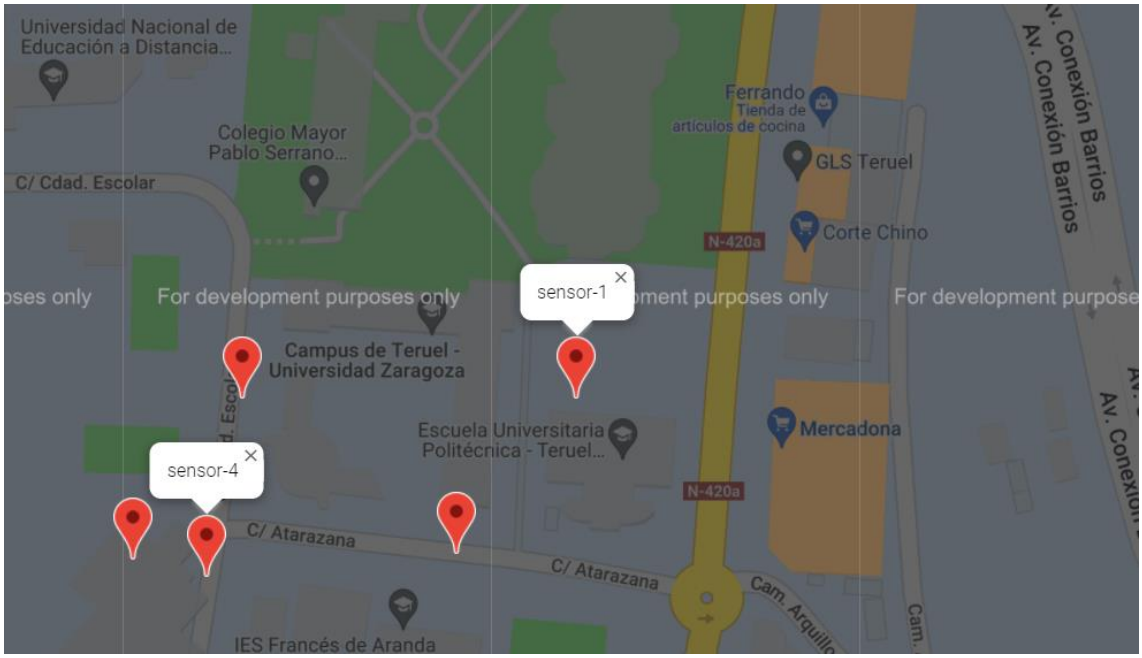

Figura 47. Ejemplo de mapa en Chirpcontrol.

5.3.4 Interfaz web de Chirpcontrol.

En la última fase del proyecto a Chirpcontrol se le añadió una interfaz web que permite remplazar la aplicación móvil ChirpRegister. El diseño de esta interfaz web es simple y permite a aquellos usuarios que no dispongan de los requerimientos para instalar ChirpRegister en su dispositivo móvil poder también realizar las tareas necesarias para dar de alta un dispositivo LoRa en Chirpstack.
Algunas imágenes de esta interfaz web se muestran a continuación:

Figura 48. Menú principal de Chirpcontrol.

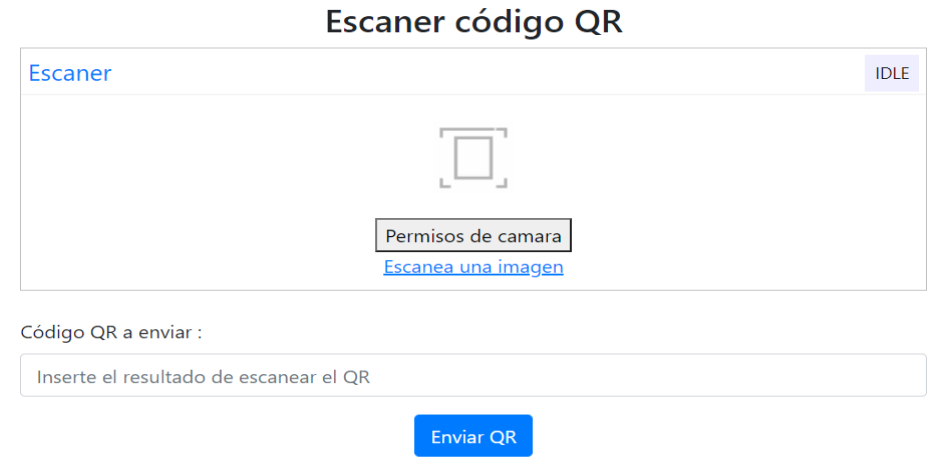

Figura 49. Escáner de códigos QR desde Chirpcontrol.

Figura 50. Consultar dispositivos registrados.

5.4. Desarrollo del firmware del dispositivo final.

Esta sección detalla los aspectos más importantes que se han tenido en cuenta a la hora de desarrollar el firmware que deben contener todos los dispositivos finales LoRA que en un supuesto caso requieran ser utilizados en un despliegue masivo.

El firmware desarrollado está diseñado para dispositivos cubecell AB02, en caso de utilizar otras clases de dispositivos que no sean de este fabricante habría que realizar adaptaciones en el código, además de tener en cuenta que todos los dispositivos LoRa empleados deben contener una pantalla ya sea externa o integrada con el propio dispositivo.

El objetivo básico del firmware desarrollado consiste en aprovechar que el dispositivo dispone de una pantalla OLED para mostrar en esta el DevEUI y el AppKey del dispositivo, ambos representados en formato código QR. Hay que tener en cuenta que dadas las reducidas dimensiones de la pantalla del dispositivo cubecell AB02 (128 x 64 pixeles) el tamaño del código QR está limitado. Habiendo realizado varias pruebas se concluye que un código QR con este tamaño puede representar como máximo un total de 54 caracteres. Teniendo en cuenta que el DevEUI consta de 16 caracteres y el AppKey se compone de otros 32 la mejor solución para poder almacenar ambos parámetros en el código QR es separarlos por un delimitador que en este caso es un punto y coma (;). De esta forma la información en el QR queda representada de la siguiente forma DevEUI;AppKey.

Un posible ejemplo es: 11CC33445566AA88;AABB3344665577EE11223344FF662244 donde los 16 primeros caracteres representan el DevEUI del dispositivo, después aparece un ; para indicar el final del DevEUI y el principio del AppKey, y finalmente aparecen 32 caracteres que equivalen al valor del AppKey.

Este código QR se ha optado por representarlo en color negro sobre fondo blanco ya que así se facilita su lectura para la mayoría de los escáneres QR y aparecerá en la pantalla del dispositivo hasta que este consiga unirse a su respectiva red LoRaWAN, después se apagara la pantalla ya que mantenerla encendida no tiene ninguna utilidad y requiere un gasto de batería que es innecesario.

Para asignar el DevEUI al dispositivo se utiliza la instrucción *LoRaWAN.generateDeveuiByChipID()* que permite obtener un DevEUI para el dispositivo según el chip que ha instalado el fabricante en este, lo que garantiza que sea un valor único y permanente. Para asignar el AppKey al dispositivo se ha diseñado un método que permite generar un valor aleatorio. Después este AppKey será almacenado en la memoria EEPROM del dispositivo junto con el periodo de transmisión.

El periodo de transmisión es la cantidad de tiempo que debe pasar entre cada transmisión de información del dispositivo final a su servidor, es decir, cada cuanto tiempo debe realizarse un mensaje uplink. El periodo de transmisión tiene un valor inicial aunque este valor puede modificarse posteriormente.

La memoria EEPROM, del dispositivo siguiendo las recomendaciones del fabricante, se le ha reservado un espacio de 512 bytes, este espacio es utilizado para almacenar tanto el AppKey como el periodo de transmisión. Estos valores almacenados en la EEPROM solo serán modificados cuando los usuarios hagan una solicitud para ello, en caso contrario, su valor siempre será el mismo.

Finalmente, para facilitar el manejo de estos dispositivos finales se ha añadido un método que permite modificar valores del dispositivo sin necesidad de actualizar su firmware, por tanto, los usuarios a distancia pueden mandar comandos directamente a los dispositivos a través de mensajes downlink para cambiar valores en estos.
Los comandos disponibles para actualizar el dispositivo son los siguientes:

- **R:** Reinicia el dispositivo.
- **C:** Borra todo el contenido de la memoria EEPROM y reinicia el dispositivo, por tanto, se genera un nuevo AppKey aleatorio y el periodo de transmisión vuelve a su valor por defecto.
- **S'duracion':** Modifica el periodo de transmisión del dispositivo asignándole la duración enviada (en segundos). El valor enviado debe estar entre 5 segundos y 12 horas. Por ejemplo: S60 provoca que se envié un mensaje uplink cada minuto.
- **T'duracion':** Funciona exactamente igual que el comando anterior, pero el nuevo periodo de transmisión se almacena en la memoria EEPROM de forma que si se reinicia el dispositivo el periodo de transmisión será el valor indicado en el último comando de este tipo enviado.

A continuación, se muestra el procedimiento con imágenes que sigue el dispositivo cubecell AB02 para unirse a una red LoRaWAN.

1. Muestra en la pantalla OLED un código QR que representa el DevEUI y el AppKey, al mismo tiempo intenta constantemente unirse a una red LoRaWAN

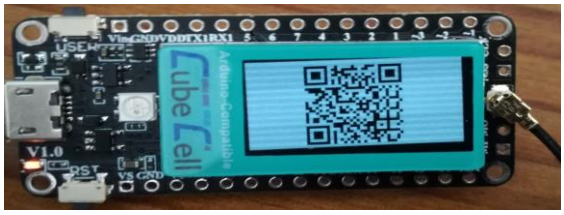

Figura 51. Se muestra un código QR en la pantalla.

2. El dispositivo consigue unirse a una red LoRaWAN y muestra un mensaje de confirmación.

Figura 52. Mensaje de confirmación.

3. El dispositivo tras conseguir unirse a la red LoRaWAN y mostrar el mensaje de confirmación apaga su pantalla para ahorrar en consumo de batería. A partir de este momento empezara a intercambiar datos con la red según este programado.

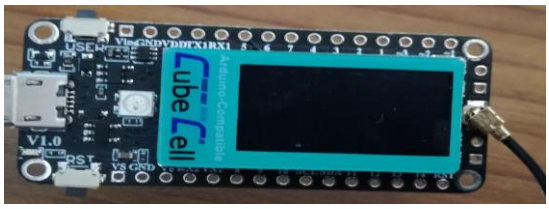

Figura 53. Apagado de pantalla para ahorrar batería.

5.5. Desarrollo de ChirpRegister (APP Android)

ChirpRegister es una aplicación desarrollada para teléfonos móvil Android cuya función principal es realizar la lectura de los códigos QR mostrados en los dispositivos finales LoRa, obteniendo así el DevEUI y el AppKey representados en ese código QR para permitir al usuario crear un nuevo dispositivo en Chirpstack con estos datos. Posteriormente ese dispositivo creado en Chirpstack será asociado con el dispositivo físico LoRa, lo que permite que este ya tenga la capacidad para empezar a transmitir datos por la red.

Como objetivos secundarios ChirpRegister trata de facilitar la gestión de los dispositivos ya creados en Chirpstack permitiendo su eliminación y actualización, además de permitir consultar de forma rápida el mapa con la ubicación de cada dispositivo.

Dejando en un segundo plano el diseño de la interfaz gráfica de la aplicación, el desarrollo de ChirpRegister se puede dividir en los siguientes apartados:

- **Escáner de los códigos QR**
  Para que la aplicación móvil pueda escanear los códigos QR de los dispositivos LoRa se ha optado por utilizar la librería *com.journeyapps:zxing-android-embedded:4.3.0*. Esta librería proporciona los métodos necesarios para poder habilitar la cámara externa de los dispositivos móviles y utilizarla como escáner QR.

  Una vez se obtiene el contenido del código QR se verifica si este cumple con el formato esperado, es decir, 16 caracteres hexadecimales seguidos de ; y este seguido de otros 32 caracteres hexadecimales. Si no se cumple esta condición se muestra un error, en caso contrario, se analiza el contenido del QR y se obtiene por un lado el DevEUI y por otro lado el AppKey. Una vez ya se dispone de estos valores el usuario simplemente tiene que asignar un nombre y una descripción al dispositivo para poder darlo de alta en Chirpstack.

- **Ubicación GPS del móvil**
  La localización del dispositivo móvil con el que el usuario está realizando el registro de los dispositivos LoRa en Chirpstack se obtiene mediante el uso de la librería *com.google.android.gms:play-services-location:18.0.0*. Esta librería de Google permite obtener las coordenadas de latitud y longitud del dispositivo móvil siempre y cuando este tenga activado el sistema GPS.

  La obtención de las coordenadas de ubicación se realiza después de que el usuario haya conseguido escanear con éxito el código QR y es opcional ya que si no es

necesario conocer las ubicaciones de los dispositivos LoRa empleados este apartado puede omitirse.

- **Comunicación con Chirpcontrol**
  Para poder completar el registro de un dispositivo en Chirpstack la app móvil ChirpRegister, tras haberse realizado la lectura de un código QR y haber obtenido las coordenadas de la localización actual, debe enviar los datos a Chirpcontrol.

  La forma de comunicación entre ambos sistemas es gracias a la API REST de la que dispone Chirpcontrol. Para poder consumir una API REST de forma simple y eficaz desde una aplicación Android se ha utilizado la librería *com.squareup.retrofit2:retrofit:2.7.2*. Esta librería ha resultado ser la más útil y eficaz frente a otras librerías probadas en este proyecto como Volley o la propia acción de ejecutar la comunicación mediante código a través de la creación de hilos que actúen en un segundo plano para consumir la API REST. Gracias a la librería *retrofit* se ha podido consumir la API REST utilizando todos los métodos que han sido necesarios, ya que ChirpRegister aparte de poder registrar nuevos dispositivos también permite la obtención, actualización y eliminación de estos, por tanto, se han utilizado peticiones Post, Put, Get y Delete.

A continuación, se muestran algunas imágenes de las pantallas más importantes de la aplicación ChirpRegister, indicando las pantallas que se siguen para registrar un nuevo dispositivo en Chirpstack.

Figura 54. Pantalla de inicio.

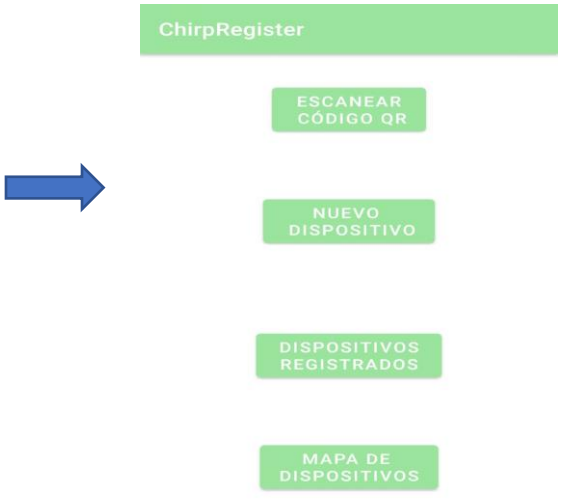

Figura 55. Menú principal.

Una vez en el menú principal, seleccionando el botón ESCANEAR CÓDIGO QR aparecen las siguientes pantallas que permiten registrar un nuevo dispositivo.

Figura 56. Escáner de Código QR.

1. Se utiliza la cámara externa como escáner de códigos QR.

Una vez el código QR se haya podido leer correctamente, si este cumple con el formato esperado, se pasa a la siguiente pantalla en la que se muestra un formulario donde ya estarán completos los campos DevEUI y AppKey, ya que estos se completan automáticamente tras leerse del código QR.

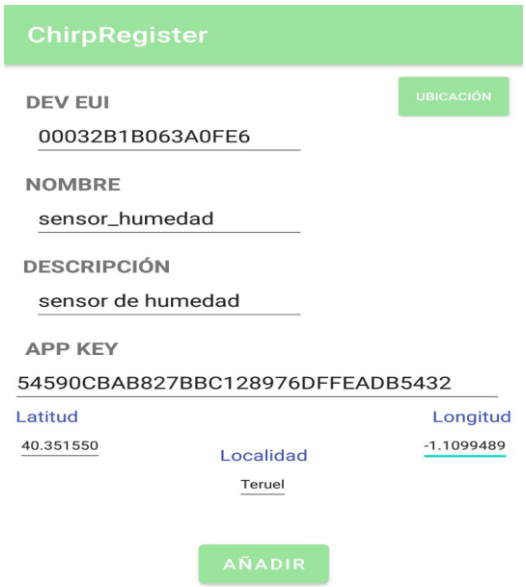

Figura 57. Formulario con la información del nuevo dispositivo a registrar.

2. En este formulario el usuario debe rellenar los campos nombre y descripción (DEV EUI y APP KEY ya estarán completos tras leerse del QR).
El usuario también puede optar por activar la ubicación para que aparezcan los campos Latitud, Longitud y Localidad.

Tras tener todos los campos del formulario completos al pulsar el botón AÑADIR el dispositivo se registrará en Chirpstack.

Si en el menú principal se selecciona el botón DISPOSITIVOS REGISTRADOS se obtienen todos los dispositivos que hay registrados y si se selecciona uno en concreto se permite la actualización y eliminación de este.

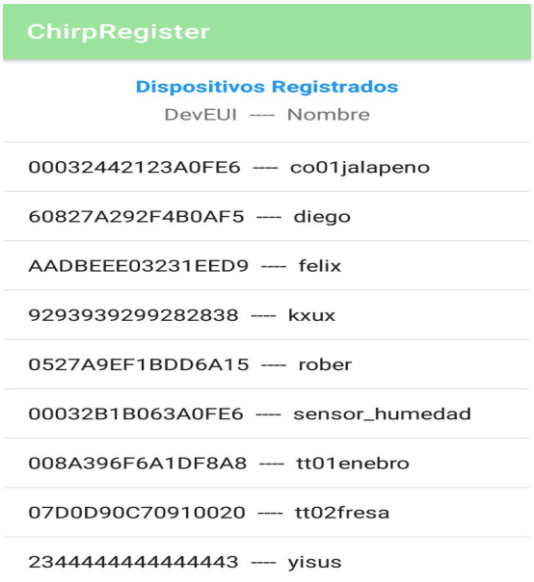

Figura 58. Lista de dispositivos registrados.

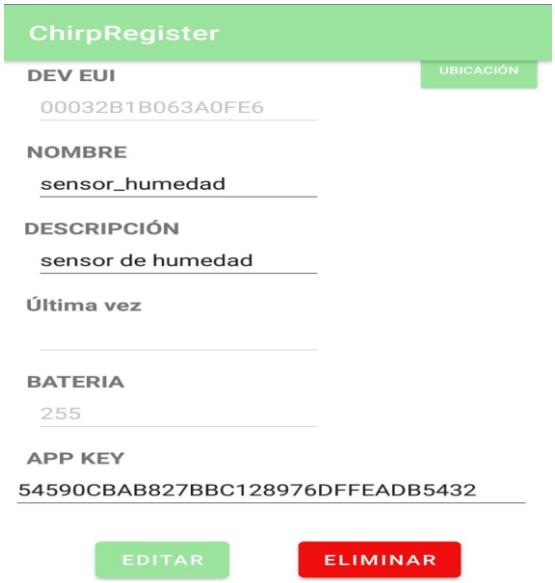

Figura 59. Información en detalle de un dispositivo.

Finalmente, los otros dos botones restantes del menú actúan de la siguiente forma:
-NUEVO DISPOSITIVO: permite registrar un dispositivo sin disponer de un código QR por lo que el usuario debe rellenar manualmente los campos DEV EUI y APP KEY.

- MAPA DE DISPOSITIVOS: Abre en el navegador un mapa donde se puede consultar la ubicación de cada dispositivo.

6. Conclusión y futuras líneas de investigación

6.1 Conclusión

En este proyecto se ha llevado a cabo el diseño y la implementación de una arquitectura software que permite incrementar la rapidez, sencillez y eficacia del procedimiento que es necesario realizar para registrar dispositivos LoRa en su respectivo sistema de gestión/explotación (este sistema de gestión es Chirpstack), automatizándose todo este proceso.

Esta arquitectura software se compone de cuatro elementos diferentes, por tanto, para su desarrollo ha sido necesario trabajar con diferentes tecnologías como Android, Arduino, web services REST, HTML, Jakarta EE, JavaScript, etc... Las funcionalidades principales de estos cuatro elementos se pueden resumir muy brevemente en las siguientes líneas.

- Programa Arduino que permite representar mediante códigos QR información en las pantallas OLED de los dispositivos LoRa e intentar establecer el registro de los dispositivos en una red LoRaWAN.
- Aplicación móvil Android que permite realizar una correcta lectura de los códigos QR y obtener la ubicación actual del dispositivo.
- Aplicación web que actúa como intermediario facilitando la comunicación entre la aplicación móvil y el sistema de gestión de los dispositivos LoRa. Proporciona una API REST que es consumida por la aplicación móvil y dispone de un mapa en el que se puede encontrar la ubicación de cada dispositivo LoRa registrado.
- API de Chirpstack que facilita la comunicación de este sistema de gestión de dispositivos LoRA con sistemas externos. Esta API es consumida por la aplicación web anterior.

En cuanto a los objetivos del proyecto, se han logrado todos y cada uno de ellos, ya que se ha diseñado y desarrollado la arquitectura software ya comentada que permite registrar la identidad única y la localización de cada dispositivo en su sistema de gestión/explotación asociado de una forma eficaz y automatizada, además se han realizado varios experimentos que demuestran que la arquitectura software funciona correctamente según lo esperado.

Respecto a la temática del proyecto, ha resultado interesante conocer y trabajar con la tecnología LoRaWAN, una nueva tecnología que presenta un gran potencial en el mundo IoT y más teniendo en cuenta el auge que se espera en los próximos años del Internet de las Cosas. Además de profundizar el conocimiento en otras tecnologías como Arduino, el desarrollo de aplicaciones Android o la importancia en el momento actual de las API.

6.2 Líneas futuras de investigación

Una propuesta de futuro para este proyecto sería incluir en cada dispositivo final LoRa un mecanismo GPS que obtenga las coordenadas de ubicación del dispositivo. Cuando un dispositivo final vaya a trasmitir información al servidor se incluiría en el mensaje enviado las coordenadas de ubicación del dispositivo y se actualizaría automáticamente la localización de este en el mapa de dispositivos. Aunque esto requeriría un precio de coste y un consumo de batería más elevado, la localización de los dispositivos estaría constantemente actualizada y no solo se conocería la localización del dispositivo en el momento en el que se une a una red LoRaWAN, como ocurre actualmente. En casos en los que los nodos finales se encuentren siempre en una situación estática no tendría utilidad, sin embargo, si se requiere que estos nodos estén en constante movimiento se podría conocer su ubicación exacta en cada momento y desde que lugar han recogido los datos enviados.

7. Bibliografía

[1] - Alonso, R., 2021. *¿Qué es el Internet de las cosas (IoT) y por qué se le llama así?*. HardZone. Disponible en: <https://hardzone.es/reportajes/que-es/internet-cosas-iot/> [Consultado el 15 de enero de 2022].

[2] - 2019. *what-is-iot*. Disponible en: <https://www.redhat.com/es/topics/internet-of-things/what-is-iot> [Consultado el 15 de enero de 2022].

[3] - Locke, J., 2021. *Selección de una plataforma de gestión de dispositivos IoT - con ejemplos de implementación*. Es.digi.com. Disponible en: <https://es.digi.com/blog/post/selecting-an-iot-device-management-platform> [Consultado el 18 de enero de 2022].

[4] - Moya Quimbita, M., 2018. *Evaluación de pasarela LoRa/LoRaWAN en entornos urbanos* Riunet.upv.es. Disponible en: <https://riunet.upv.es/bitstream/handle/10251/109791/Moya%20-%20Evaluaci%C3%B3n%20de%20pasarela%20LoRa/LoRaWAN%20en%20entornos%20urbanos.pdf?sequence =1&isAllowed=y> [Consultado el 18 de enero de 2022].

[5] - Marín Cava, M., 2020. *Sistema de monitorización para elementos móviles IoT mediante redes lpwan*. Repositorio.upct.es. Disponible en: <https://repositorio.upct.es/xmlui/bitstream/handle/10317/8861/tfg-mar-sis.pdf?sequence=1&isAllowed=y> [Consultado el 20 de enero de 2022].

[6] - Es.wikipedia.org. 2022. *LoRaWAN - Wikipedia, la enciclopedia libre*. Disponible en: <https://es.wikipedia.org/wiki/LoRaWAN> [Consultado el 21 de enero de 2022].

[7] - 2022. *What Is LoRa®?* Disponible en: <https://www.semtech.com/lora/what-is-lora> [Consultado el 21 de enero de 2022].

[8] - Lora-developers.semtech.com. 2022. *LoRa y LoRaWAN: Resumen técnico | PORTAL DEL DESARROLLADOR* . Disponible en: <https://lora-developers.semtech.com/documentation/tech-papers-and-guides/lora-and-lorawan/> [Consultado el 21 de enero de 2022].

[9] - Pérez, E., 2018. *LoRaWAN*. Medio. Disponible en: <https://medium.com/pruebas-de-laboratorio-de-la-modulaci%C3%B3n-lora/lorawan-d00f48384160> [Consultado el 21 de enero de 2022].

[10] - Yuri, 2021. *chirpstack » El Laboratorio del Dr. Yuri* . Bitácora.eniac2000.com. Disponible en: <https://bitacora.eniac2000.com/?tag=chirpstack> [Consultado el 23 de enero de 2022].

[11] - BConsultores. 2020. *ChirpStack tu servidor de red LoraWan privado - BConsultors*. Disponible en: <https://bconsultors.com/2020/12/10/chirpstack-tu-servidor-de-red-lorawan-privado/> [Consultado el 24 de enero de 2022].

[12] - Programmerclick.com. 2019. *Tutorial de conexión del enlace completo LoRa 04: uso de ChirpStack para crear un servidor LoRa LAN privado en Windows local - programador clic*. Disponible en: <https://programmerclick.com/article/5376680779/> [Consultado el 24 de enero de 2022].

[13] - AlfaIOT. 2022. *Chirpstack*. Disponible en: <https://alfaiot.com/tecnologias-iot/lorawan/chirpstack/> [Consultado el 24 de enero de 2022].

[14] - Heltec Automatización. 2022. *CubeCell – Placa de desarrollo Plus*. Disponible en: <https://heltec.org/project/htcc-ab02/> [Consultado el 2 de febrero de 2022].

[15] - GitHub. *NO se supone que un LoRaWAN DevAddr sea único · Problema n.º 10 · IOActive/laf*. Disponible en: <https://github.com/IOActive/laf/issues/10> [Consultado el 2 de febrero de 2022].

[16] - Sabas, 2017. *Haciendo IoT con LoRa: Capítulo 2.- Tipos y Clases de Nodos* . Medio. Disponible en: <https://medium.com/beelan/haciendo-iot-con-lora-capitulo-2-tipos-y-clases-de-nodos-3856aba0e5be> [Consultado el 6 de febrero de 2022].

[17] - Static1.squarespace.com. *LoRaWAN - OTA or ABP?* Disponible en: <https://static1.squarespace.com/static/560cc2c2e4b01e842d9fac18/t/5a938d38ec212d9451fbecf8/1519619387035/OTAA_or_ABPv3.pdf> [Consultado el 6 de febrero de 2022].

[18] - RAKwireless Store. 2022. *WisGate Edge Lite | RAK7258 | Puerta de enlace para LoRaWAN* . Disponible en: <https://store.rakwireless.com/collections/wisgate-edge/products/rak7258-micro-gateway?variant=39942876430534> [Consultado el 13 de marzo de 2022].