Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

IoT para la sensorización ambiental de dependencias de la ULL

IoT sensoring for environmental data of the ULL dependencies

Thaddäus Haase

La Laguna, 8 de julio de 2022

D. **Alberto Francisco Hamilton Castro**, con N.I.F. 43.773.884-P profesor Titular de Universidad adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutor

D. **Jesús Miguel Torres Jorge**, con N.I.F. 43.826.207-Y profesor Contratado Doctor adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como cotutor

C E R T I F I C A (N)

Que la presente memoria titulada:

”*IoT para la sensorización ambiental de dependencias de la ULL*”

ha sido realizada bajo su dirección por D. **Thaddäus Haase**, con N.I.E. X9777504C.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos, firman la presente en La Laguna a 8 de julio de 2022

Agradecimientos

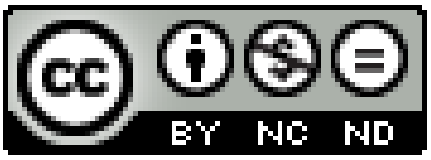
En primer lugar, me gustaría agradecer a mi familia, por apoyarme siempre a lo largo de estos años.

También, agradecer a mis tutores, Alberto y Jesús, por darme la oportunidad de realizar este proyecto y guiarme durante su desarrollo. En tercer lugar, asimismo agradecer a Cátedra Cajasiete Big Data, Open Data y Blockchain por financiar este proyecto brindando los dispositivos que contienen los sensores a la Universidad de La Laguna.

Y por último, a los amigos que he conocido durante la universidad, por ayudarme a despejar mi mente y a relajarme en los momentos más estresantes de la carrera.

Sin duda, sin alguno de ellos, este proyecto hubiera sido mucho más complicado.

Licencia



Resumen

Hoy en día, estamos viviendo un auge en las tecnologías de Internet of Things y un importante avance en las tecnologías de comunicaciones. A pesar de la pandemia y de las guerras actuales, los dispositivos electrónicos como los microchips están, por lo general, disminuyendo tanto en su tamaño como en su precio. Todo esto beneficia la evolución constante del IoT. Además, tanto el ahorro de energía como el control de las constantes ambientales, cobra cada vez mayor importancia por los problemas derivados del cambio climático y se está empezando a tomar consciencia del peso que tienen estos dos factores en la esperanza de vida de la población.

Por otro lado, para llevar a cabo la conexión entre los sensores disponemos de diversos estándares de comunicación. Entre ellos destacaremos el protocolo LoRaWan por sus características de bajo consumo y distancia media de comunicación. También se harán comparaciones entre las diferentes versiones que posee, con sus ventajas y desventajas.

El presente proyecto propone una solución a pequeña escala, con la finalidad de monitorizar, supervisar y alertar de las constantes ambientales de las diferentes dependencias de la ULL.

Palabras clave: Internet de las cosas, Internet of Things, Universidad de La Laguna, The Things Network, LoRa, LoRaWan, Sensorización ambiental.

Abstract

Currently, we are witnessing a boom inside the technology sector of the Internet of Things, including advances in communications technologies. Despite the current pandemic situation and the ongoing war, electronics are getting smaller and, normally, cheaper to produce and to acquire. Those factors have a positive impact in the development of IoT. In addition, climate change and the ongoing pandemie give new importance to saving energy and controlling environmental factors, which in turn, raises awareness of how important these factors are to the well being of the population.

To enable the implementation of IoT, we also have to take a look at the different communication standards that enable the connection between sensors. Among them we will highlight the LoRaWan protocol due to its characteristics of low energy consumption and its medium to long range communication distance. Comparisons will also be made between the different versions it has, with their advantages and disadvantages.

This project proposes a small-scale solution, with the purpose of monitoring, supervising and if necessary, alerting of the environmental factors inside of the different dependencies of the ULL.

Keywords: Internet of Things, Universidad de La Laguna, The Things Network, LoRa, LoRaWan, Environmental sensorization.

Índice general

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

El objetivo del proyecto es estudiar posibles opciones de sensorización de diferentes parámetros ambientales (temperatura, humedad, concentración de CO2, etc.) en diferentes dependencias de la ULL (laboratorios, aulas, pasillos, etc.). Esto se consigue colocando sensores adecuados en puntos estratégicos de dichas dependencias y conectándolos a Internet, de manera que se pueda presentar la información recogida de forma local o centralizada. Este tipo de soluciones entra dentro del campo de lo que se conoce como Internet de las Cosas (Internet of Things → IoT).

1.1. Definición del problema

Durante estos dos últimos años, debido a la pandemia que hemos sufrido, el control de la calidad del aire y del aforo de las personas en los establecimientos ha adquirido una nueva dimensión de importancia en la sociedad. Pero, no solo es importante el control de la calidad del aire para evitar contagios, sino que también este es un factor muy relacionado con la contaminación y con la esperanza de vida de las personas. Aquí sobre todo el nivel de CO2 y el TVOC[1] son factores con los que podemos estimar bien la calidad del aire. Otros factores ambientales que pasan algo más desapercibidos, pero que son igualmente importantes, podrían ser la temperatura, la humedad o incluso la luminosidad.

1.2. Justificación

Con el presente trabajo se ha estudiado la posibilidad de aplicar tecnologías IoT para llevar una monitorización exhaustiva de las condiciones ambientales de las instalaciones de la Universidad de La Laguna. Llevar a cabo tal control podría suponer poder avisar a los usuarios cuando se encuentren en peligro debido a las condiciones del aire, previniendo situaciones que puedan afectar a la vida de las personas. Además, se puede llevar a cabo un control de aforo utilizando el contador de personas que entran y salen de un establecimiento. Este podría tener beneficios a la hora de gestionar la logística del lugar y no sobresaturar las instalaciones.

Igualmente, este proyecto también podría ayudar a la lucha por el correcto uso de los recursos disponibles, ya que al monitorizar temperatura, humedad etc., se podría controlar mejor la cantidad de energía utilizada en sistemas de calefacción o de refrigeración.

[1] **TVOC**: (Total Volatile Organic Compounds) mide el total de los compuestos orgánicos volátiles suspendidos en el aire

Por último, en los últimos años el IoT no ha parado de crecer, incluso teniendo problemas globales que han afectado fuertemente a algunos sectores. Gracias a esta tendencia de crecimiento positiva, es de esperar que este tipo de tecnologías sean cada vez más comunes en la vida cotidiana de las personas. Por lo cual, este proyecto podría ayudar a la Universidad a mantenerse en las últimas tendencias y tecnologías, modernizando, aún más, sus servicios.

1.3. Tendencia de mercado

Por lo general, las tendencias de mercado para el IoT han crecido consistentemente en los últimos años hasta llegar a los años de la pandemia y de la guerra. Estas situaciones excepcionales han afectado considerablemente al comercio y a la producción de microprocesadores. Por estos factores ajenos a la tecnología, su expansión se ha visto mermada en los dos últimos años pero, aun así, ha continuado con un crecimiento constante.

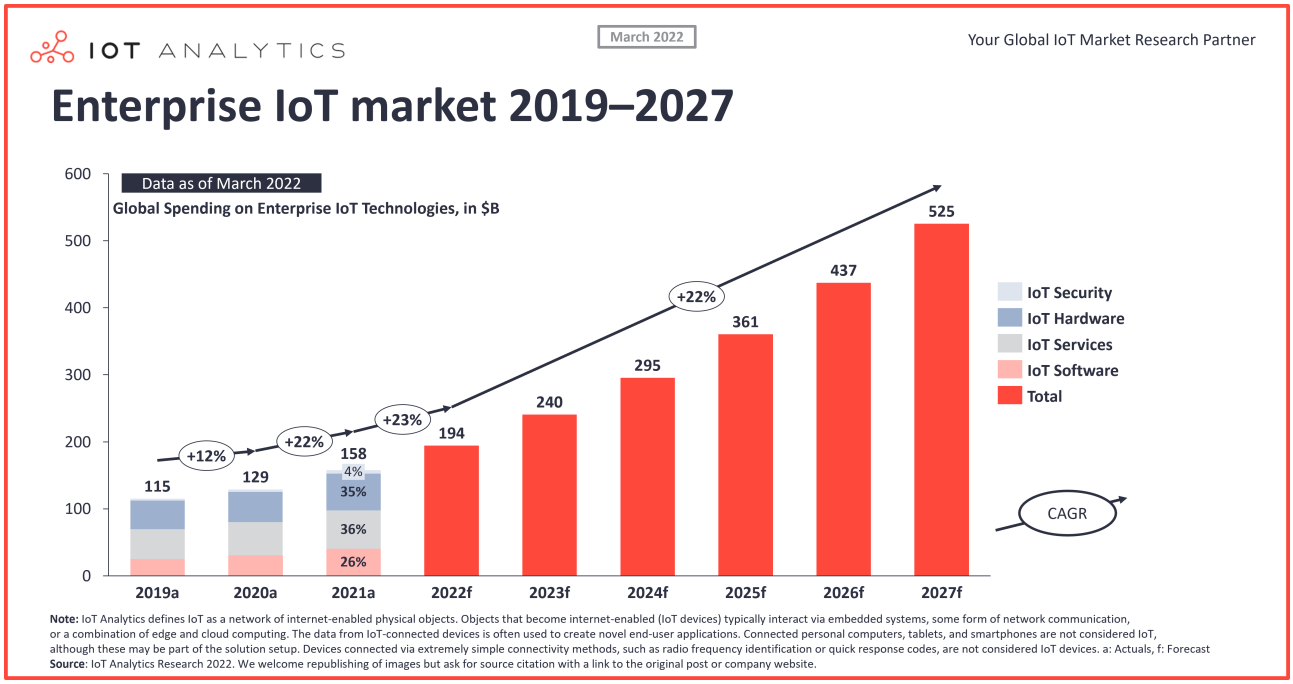


Figura 1.1: Estado actual y predicción del mercado de IoT [1].

Si deseamos hacer un análisis más profundo, podemos consultar los diferentes factores que influencian el mercado de IoT [1].

- **Factores económicos**: en estos momentos nos encontramos ante una situación del mercado un poco difícil, sobre todo, por los efectos de la inflación. Esta inflación hace que las empresas no dispongan de tanto dinero para invertir en nuevas tecnologías. Esto, sumado a la carencia de microprocesadores, causada por los efectos económicos de Covid-19, hace que el mercado de IoT esté creciendo menos de lo esperado. Pero, incluso en este panorama tan desfavorable, IoT tiene la posibilidad de destacar por las numerosas ventajas que ofrece. Por estas razones, hay empresas que siguen invirtiendo en IoT aunque en versiones más limitadas.

- **Políticos y sociales**: lamentablemente, la política puede afectar mucho al mercado. Debido a la situación actual, y teniendo en cuenta que Rusia es uno de los exportadores de materias primas necesarios para la producción de microchips, es muy

probable que la situación del mercado se complique aún más en los próximos meses. Pero la nueva Ley de Datos propuesta por la Unión Europea (EU del inglés European Union) [2] facilita la implementación de nuevas tecnologías y, por lo tanto, también afecta al IoT.

- **Medio ambiente**: para limitar los efectos del cambio climático, están intentando reducir al máximo las emisiones de CO2. Es aquí donde IoT puede ayudar en el control de energía, y en general, de los recursos. En suma, podemos afirmar que la intención de minimizar el cambio climático, es uno de los factores que más favorece la expansión de los servicios de IoT.

- **Sanitarios**: por último, se suma tanto la situación anómala como el miedo de posibles futuras pandemias que ha generado el Covid-19. Debido a la nueva situación sanitaria el interés de monitorear ciertos espacios aumenta. Sobre todo, el interés de monitorizar la calidad del aire ha adquirido un nuevo nivel de importancia en nuestro día a día.

Podemos observar que hay algunos puntos que favorecen el crecimiento del mercado IoT y otros que no. Pero debemos destacar que, incluso con todos los factores perjudiciales actuales, se espera que el mercado de IoT obtenga un crecimiento de entorno a un 22 % durante los siguientes años. Esta cifra es algo inferior a la que se esperaba en 2021, la cual estaba en torno a un 24 % [3], pero teniendo en cuenta todos los factores adversos, que solo haya bajado un 2 % del crecimiento estimado sitúa al IoT como una tecnología con potencial hacia el futuro.

1.4. Objetivos

Una parte muy importante del trabajo será el estudio de dispositivos de bajo consumo para el IoT. Como ya hemos mostrado en este estudio, la tendencia del mercado va hacia implementar cada vez más sensores para así capturar más información del entorno. El segundo punto más importante es la comunicación para poder enviar estos datos hacia un servidor, donde pueden ser guardados, visualizados o utilizados con fines estadísticos o analíticos.

Para la realización de este trabajo se han definido los siguientes objetivos:

- Estudiar los sensores disponibles en el mercado para determinar los más adecuados para la sensorización IoT. Se debe dar importancia a los que incluyen tecnologías que permiten alimentarlos a través de baterías.

- Estudiar los posibles gestores disponibles para el IoT.

- Crear una aplicación para guardar los datos recibidos y poder visualizarlos

    - Diseñar un modelo de base de datos que permita, al menos, almacenar los datos recibidos desde los sensores.
    - Desarrollar un backend que permite recibir y procesar los datos, y permitir el acceso a los mismos a través de una interfaz.
    - Diseñar e implementar un frontend que permite, al menos, la visualización de los datos.

    - Establecer un sistema de comunicación entre todos los módulos desarrollados.
    - Implementar un sistema de alertas y comunicación de las mismas a los usuarios.
- Hacer pruebas de los sensores para comprobar el correcto funcionamiento de la aplicación y de la comunicación.

Capítulo 2

Estudio previo

Uno de los primeros pasos de nuevos proyectos es, sin duda, el estudio de las diferentes herramientas, tecnologías y dispositivos existentes en el mercado actual para facilitar su implementación. En esta sección trataremos los dispositivos y tecnologías encontrados y sobre los cuales se va a llevar a cabo el proyecto.

2.1. Internet of Things

Existen varias plataformas y estándares de comunicación para añadir dispositivos al IoT. Dependiendo del caso de uso, nos pueden interesar unos u otros. Pero, para saber cual de ellos elegir, primero debemos saber que tecnología de comunicación nos conviene.

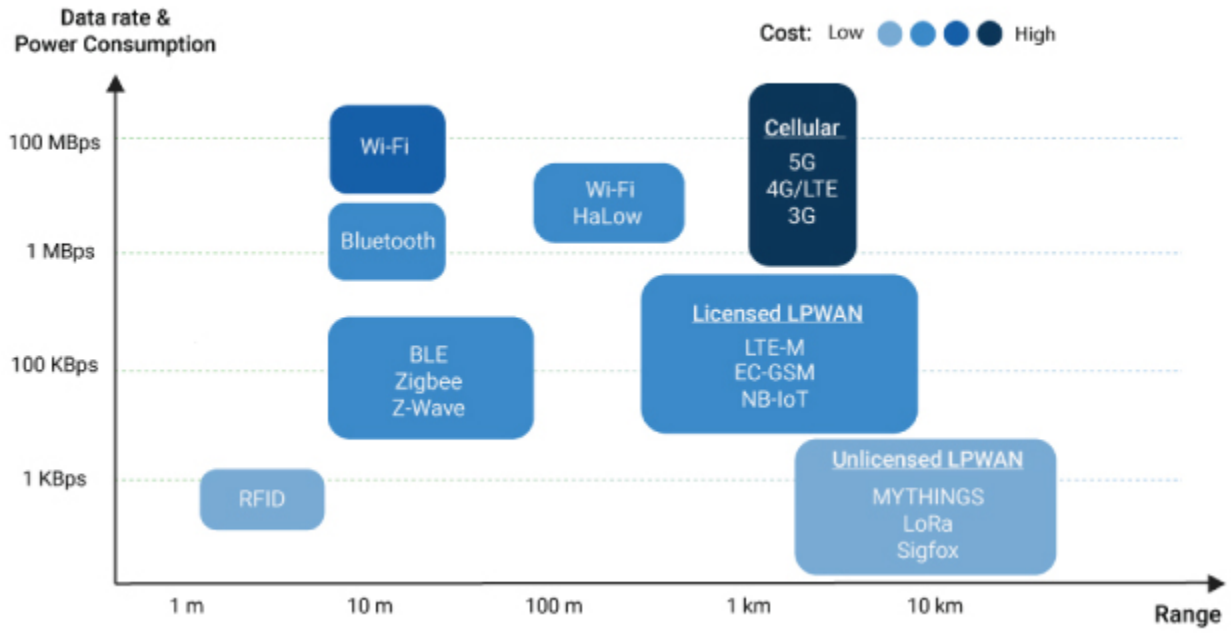


Figura 2.1: Tecnologías de comunicación inalámbricas. [4]

Como podemos observar en la imagen, existen varias tecnologías de comunicación para el IoT. Cada una de ellas tiene aspectos positivos y negativos[5], por lo cual, debemos escoger la que mejor se adapte a nuestras necesidades. En nuestro caso, vamos a monitorizar datos ambientales de la ULL, es decir, parámetros que no suelen variar abruptamente

en cortos plazos de tiempo, por lo cual, no necesitamos una comunicación lo más rápida posible. También, nos interesa que los dispositivos sean de fácil instalación, por tanto, es más conveniente utilizar aquellos que están alimentados por baterías. Así, probablemente, nos moveremos a un consumo menor pero a costa de no tener una conexión rápida. Otro aspecto a tener en cuenta es que tengan la mayor distancia posible para no tener que instalar muchas AP (Acces points) para recibir los datos.

Considerando todo, hemos decidido utilizar la tecnología LoRa[1] de los LPWAN[2] sin licencia.

Para la recogida de estos datos existen diferentes opciones en el sector. Pero, lo interesante siempre es que tengan AP ya instalados y que sean, si es posible, de acceso público. Además, sería conveniente que el usuario solo necesite tener el dispositivo final, o sensor, siempre y cuando se encuentre dentro del rango de un punto de acceso.

Para la gestión de los dispositivos y para tener acceso a la infraestructura ya disponible[7], vamos a hacer uso de The Things Network (TTN) que es básicamente una red global que utiliza LoRaWan para la comunicación y que viene con el servidor de The Things Stack (TTS) integrado. Este servidor ofrece una plataforma que nos permite gestionar nuestras aplicaciones, gateways y dispositivos finales que se conectan a TTN. Hay que tener en cuenta que tiene dos versiones; la “Community Edition”, relacionada con TTN, y la “Cloud edition”, destinada a empresas. Ambas pertenecen a The Things Industries. Sin embargo, como la Çommunity Edition"es una versión limitada y gratuita, destinada a personas que desean aprender a utilizar esta tecnología, va a ser la que utilizaremos durante este trabajo.

2.2. Sensores

Dentro de IoT existen una gran variedad de sensores para todo tipo de aplicaciones. Muchos dispositivos hacen una agrupación de sensores y añaden más funcionalidades, como, por ejemplo, una pantalla en la que las personas puedan monitorizar los datos directamente en los dispositivos. Otros vienen sueltos en formato de circuito integrado, que son destinados a aplicaciones que las integran con Raspberry Pi o Arduino.

Como vemos, existen una gran variedad de sensores para el tema de IoT, dependiendo del caso de uso nos pueden interesar unos u otros. En nuestro caso, nos interesan sensores que miden la calidad del aire y la cantidad de gente que hay en un cierto espacio.

2.2.1. Sensores adquiridos

Para la implementación del entorno de pruebas, se han adquirido los siguientes sensores.

[1] **LoRa**: (del inglés Long Range) Es una tecnología de comunicación inalámbrica basado en CSS (Chirp Spread Spectrum) y es ampliamente utilizada en los dispositivos que se comunican a través de LoRaWan. [6]

[2] **LPWAN**: (Low Power Wide Area Networks) es un protocolo de transporte inalámbrico de bajo consumo y uno de los más utilizados en el sector IoT

Milesight AM307

Es un sensor que mide la calidad de aire, luminosidad y movimiento (sensores: Temperatura, CO2, Humedad, Movimiento, Presión, TVOC). Puede operar con baterías o enchufando un cable de tipo USB-C y tiene una pantalla E-Ink para presentar los datos en local [8].

Figura 2.2: Sensor de calidad de aire AM 307.

Milesight VS 121

Es un contador de aforo que permite, no solo controlar el aforo completo, sino también dividirlo en diferentes zonas, o contar cuántas personas han pasado por una línea configurada [9].

Figura 2.3: Sensor de contador de aforos VS 121.

Dragino LHT65

También se han hecho algunas pruebas iniciales con un sensor que ya era propiedad de la ULL. Es un sensor LHT65 de la compañía Dragino, que tiene integrado un sensor de

temperatura y de humedad. Además, dispone de la posibilidad de conectar otro sensor capaz de medir la temperatura externa [10].

Figura 2.4: Sensor Dragino LHT65.

2.3. La aplicación

Para llevar a cabo el estudio previo de la aplicación, se ha tomado la decisión de dividirla en diferentes bloques, los cuales se ilustran a continuación:

2.3.1. Recogida de datos desde TTN

TTN tiene un servicio de almacenamiento en la nube [11] para los datos. Además, ofrece sistemas para integrarlo en aplicaciones propias y manejar así, el almacenamiento en nuestro propio servicio [12].

Tras el estudio inicial, se ha decidido optar por utilizar el servicio MQTT[13] para recibir los datos en un servidor local. MQTT es un protocolo de mensajería de publicación y suscripción estándar de OASIS[3] para el IoT.

2.3.2. Backend

En el backend necesitamos una API[4], una base de datos, un sistema de comunicación de alarmas (por ejemplo, Bots de Telegram) y WebSockets (WS) para el envío de mensajes en tiempo real al frontend.

Entorno de programación

La principal decisión es qué entorno de programación es más conveniente. Tenemos dos que nos ofrecen todas las funcionalidades que necesitamos. Una opción sería utilizar

[3]**OASIS**: es el acrónimo para Organization for the Advancement of Structured Information Standards. Es un consorcio internacional sin fines de lucro que se orienta al desarrollo, la convergencia y la adopción de los estándares de comercio electrónico y servicios web.

[4]**API**: Aplication Programming Interface, es una interfaz que puede considerarse como un contrato de servicio entre dos aplicaciones [14]

Python[15], existe paho-mqtt[16] para crear un cliente MQTT, tenemos a la librería FastAPI[17] para la creación de la API y también existen librerías para integrar la comunicación con diferentes gestores de bases de datos. Pero ya muchos proyectos no utilizan Python en el backend. El otro entorno es NodeJS[18] y existen algunos puntos a favor de NodeJS para esta aplicación. Uno de ellos es la velocidad. Aunque no es tan importante en una aplicación de IoT, puede ser interesante cuando el proyecto integra cada vez más dispositivos. Además, permite crear una consistencia de lenguaje, la mayoría de las aplicaciones de frontend utilizan JavaScript (o TypeScript[19]), que permite un mantenimiento más fácil ya que todo el proyecto será escrito en un único lenguaje de programación. Con respecto a las librerías, NodeJS ofrece todas las librerías que nos hacen falta para el Proyecto. Una de las más utilizadas para las API es NestJS[20]. NestJS permite crear API fácilmente escalables, ya que utiliza una separación por módulos independientes. Es open source, utiliza TypeScript y tiene una buena documentación y extensiones que permiten facilitar el mantenimiento e implementación de la API.

Comunicación

Para comunicar el frontend con el backend se utiliza la estructura de REST API[5]. Otro sistema estudiado ha sido GraphQL[22], pero debido a que la mayoría de los datos se pueden presentar como recursos, se ha optado por utilizar la REST API (CRUD[6]) típica. Aparte de esto, también se implementará un sistema de comunicación basado en WS para permitir la comunicación en tiempo real, y para que el backend pueda avisar al cliente directamente y este no tenga que comprobar si existen nuevos datos en intervalos de tiempo predefinidos. Esto permite informar a todos los usuarios conectados de nuevas alarmas o fallos del sistema.

Base de datos

Para elegir qué base de datos implementar se han estudiado dos opciones, ambas bases de datos relacionales, MariaDB[23] y PostgreSQL[24]. El punto a favor de PostgreSQL es sin duda el rendimiento, pero uno de los puntos a favor de MariaDB es que permite la replicación master-master[7]. Es más, MariaDB tiene la ventaja de que muchas veces puede tener las partes consultadas más frecuentemente ya guardados en memoria. Esto permite un mejor rendimiento en algunas consultas que requieren tags adicionales como, por ejemplo, ordenar los resultados.[25] Para añadir adaptabilidad en el futuro se utiliza TypeORM[26] para la abstracción, el cual, permite que se mueva fácilmente a otra base de datos en caso de que haga falta.

2.3.3. Frontend

Existen muchos frameworks para la parte del frontend. Los más populares son, entre otros, React[27], Vue[28] y Angular[29]. Aunque todos ofrecen sus puntos a favor, en este proyecto se ha optado por utilizar React, ya que es el más demandado y permite escribir

[5] **REST API** o API RESTful cumplen con el diseño REST o transferencia de estado representacional y reflejan los recursos, por lo cual, suele ser más fácil de escalar y mantener que otras soluciones [21]

[6] **CRUD** Siglas que hacen referencia a las acciones que ofrece: Crear, Leer, Actualizar y Borrar (del inglés: Create, Read, Update, Delete)

[7] **replicación master-master**: permite la instalación en dos o más servidores para tener redundancia y hace que la BBDD actúe como un clúster.

fácilmente componentes independientes. Además, tiene buena implementación de tipos y dispone de muchas bibliotecas que posibilitan extender las funcionalidades.

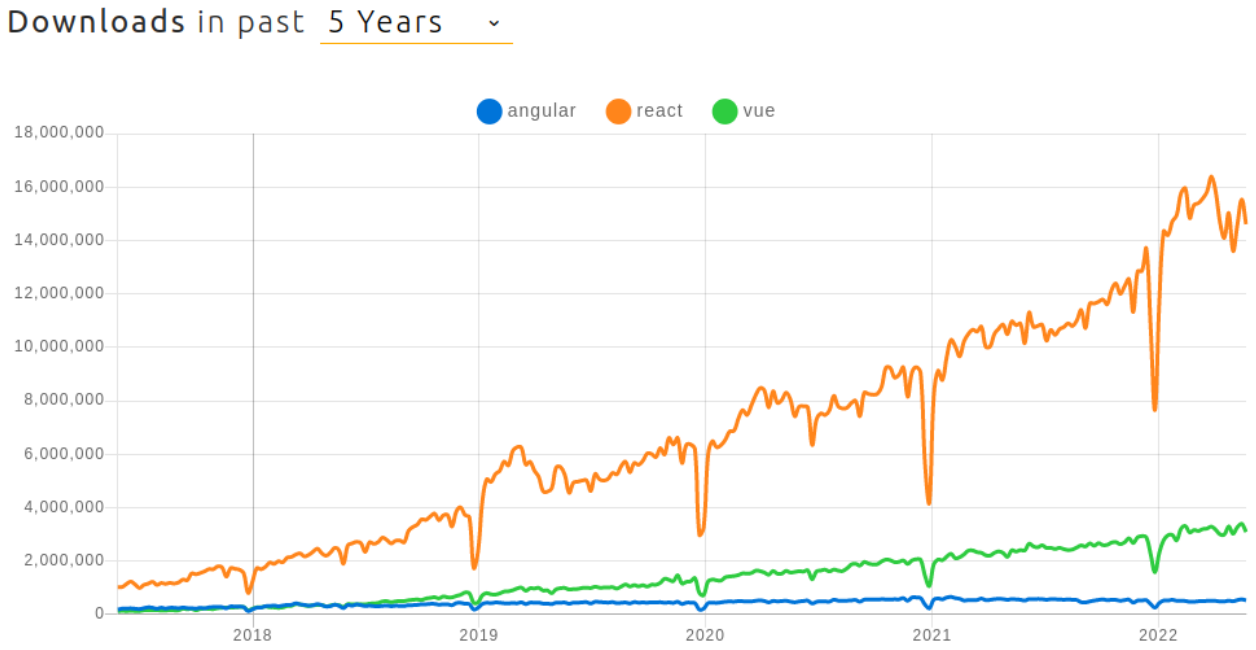


Figura 2.5: Número de descargas de los últimos 5 años de los tres frontend frameworks más populares [30].

2.3.4. Entorno y tecnologías de desarrollo

Tras llevar a cabo el estudio previo anteriormente descrito se ha decidido optar por el siguiente entorno de desarrollo:

Plataforma IoT	LoRaWan con TTN (TTS)
Comunicación	MQTT, WS, REST API
Programación	NodeJS y Visual Studio Code
Lenguaje	TypeScript
Backend	NestJS
Frontend	React
Control de versiones	Git

Tabla 2.1: Tecnologías utilizados durante el desarrollo.

Capítulo 3

Desarrollo

3.1. The Things Network

Al hablar de LoRaWan estamos haciendo referencia a la segunda capa, que es la capa MAC[1] del protocolo LoRa y que es un estándar abierto que permite el acceso a la arquitecturas LoRa [31]. Con LoRa se refiere a la capa física que es basado en la tecnología CSS (Chirp Spred Spectrum) [32] estos chirps están definidos sobre el tiempo para sincronizar la comunicación [33].

Para esta sincronización y transmisión sobre las diferentes distancias, se implementan otros protocolos que se estudiarán más adelante.

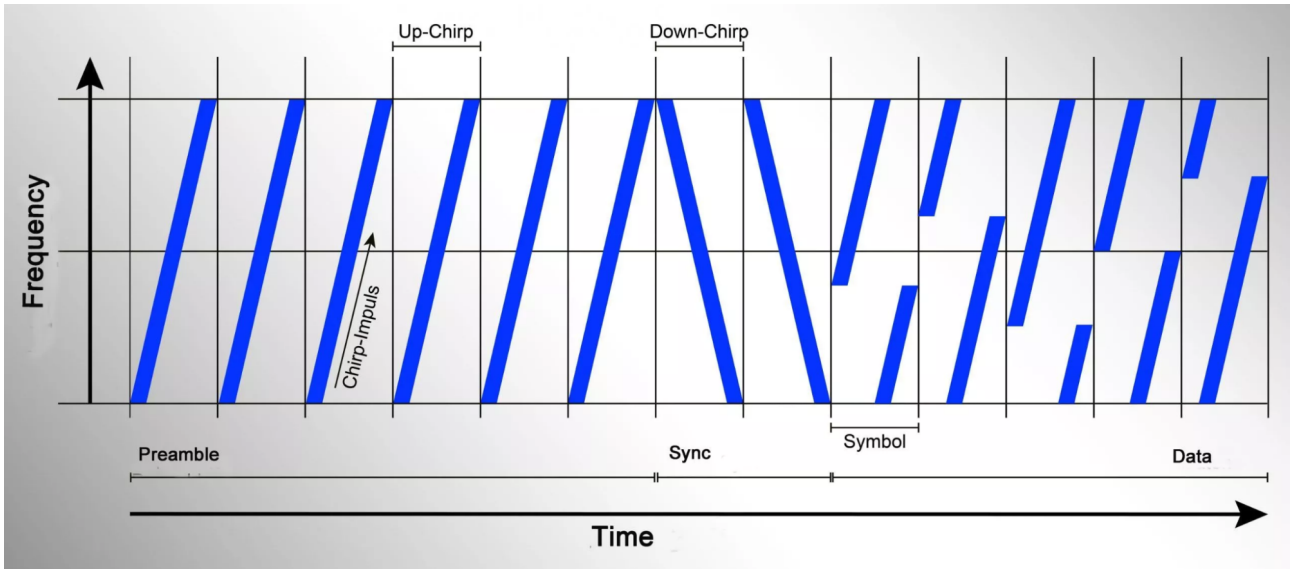


Figura 3.1: Ejemplo de la estructura de los Chirps de LoRa [34].

Para poder dar de alta los dispositivo y utilizar esta plataforma, tenemos que tener en cuenta sobre todo los siguientes puntos relacionados con LoRaWan.

3.1.1. Versiones LoRaWan

Tenemos distintas versiones de LoRaWan que siguen diferentes protocolos de comunicación. A pesar de que la versión 1.1.0 trae algunas mejoras en la seguridad y en el

[1] **capa MAC**: es la capa de controla el acceso al medio (Media Access Control)

protocolo de comunicación, en esta aplicación hemos decidido utilizar la versión 1.0.3, ya que, tras haber realizado algunas pruebas con los dispositivos, se demostró que esta versión tiene menos pérdidas de paquetes en el proceso de la comunicación. Si desea profundizar más en este aspecto, existen papers que explican la diferencia entre las versiones [35].

3.1.2. Frecuencias

Este parámetro depende mucho de donde nos encontremos. Debido a que este trabajo se ha hecho en la EU, vamos a tener en cuenta solo las frecuencias utilizables en toda la EU (pero existen países, pertenecientes a la EU, que permiten la utilización de otras frecuencias [36]). En la EU vamos a utilizar LoRaWan en las frecuencias no licenciadas EU863-870. Tenemos que tener en cuenta que estas frecuencias tienen un límite máximo de utilización (duty cycle), que en este caso es de un 1 % al día [37].

Debemos tener presente que estos son los límites puestos por la ETSI[2]. Estos coinciden con los límites de LoRaWan y los tendríamos que cumplir incluso en una red privada. Pero algunas redes, como es el caso de TTN, añaden una "Fair Use Policy" que limita más el tiempo. En TTN tenemos límites de *uplink*[3] *airtime*[4] de 30 segundos al día, mientras que los mensajes *downlink*[5] se limitan a 10 por día [38].

Es por esto que los dispositivos destinados a trabajar con LoRaWan envían los datos en bytes y con la mayor compresión factible. Para así minimizar en lo posible el *airtime*.

3.1.3. Spreading factor

Para poder controlar el *airtime* de los sensores, tenemos que tener en cuenta el *Spreading Factor* (SF) [39] de la señal, este controla el nivel de envío de los datos. Los SF disponibles son de SF7 hasta SF12. Además, debemos tener en cuenta que cada cambio de nivel duplica (aproximadamente) el *airtime* o *time-on-air*. Por lo cual, duplica el tiempo que pasa desde que se inicia la transmisión de un paquete de datos hasta que termina la transmisión. Es decir, es recomendable siempre tener el menor SF posible.

[2] **ETSI**: (European Telecommunications Standards Institute) es una organización, sin fines de lucro, que se ocupa de la normalización de estándares de la industria de las telecomunicaciones de Europa, pero que tiene proyección mundial.

[3] **uplink**: Es el mensaje que se envía desde el dispositivo hasta el AP.

[4] **airtime**: Es el tiempo en el cual el dispositivo está activamente enviando o recibiendo datos.

[5] **downlink**: Es el mensaje que se envía hacia el dispositivo.

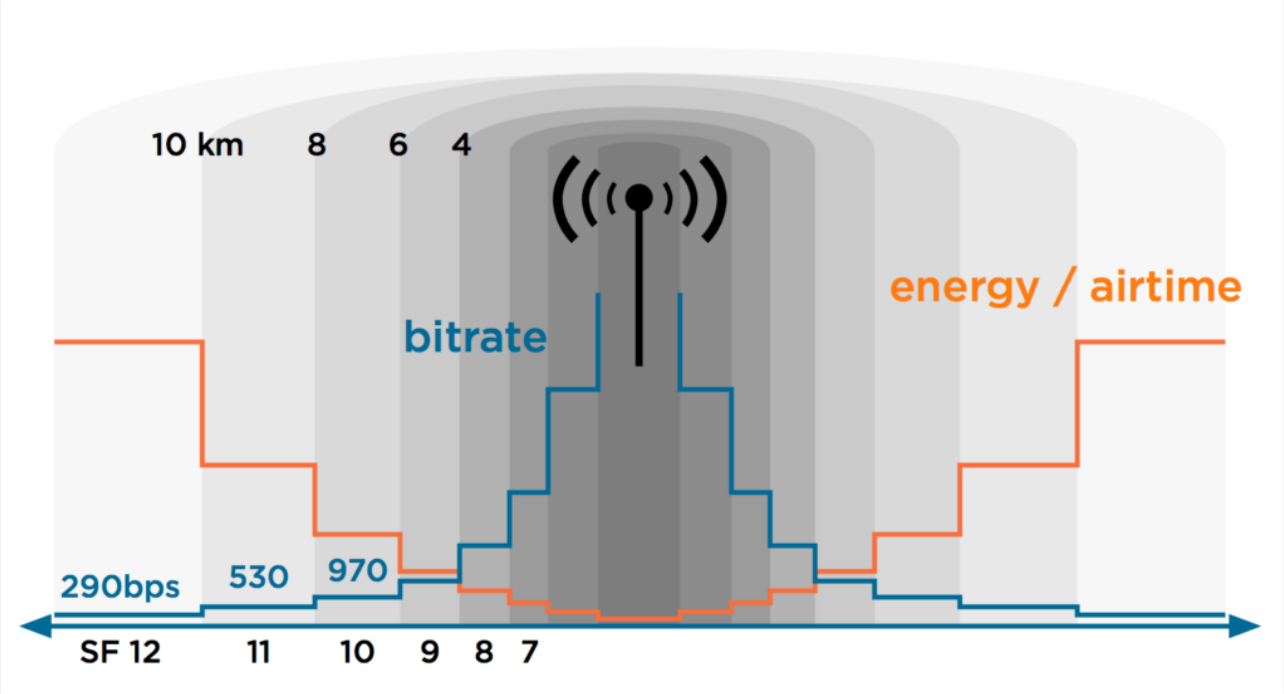


Figura 3.2: Relación entre distancia, airtime y velocidad de transmisión de los spreading factors [40]

Aumentando el SF, aumenta la distancia [41], pero también consume más energía [42]. Para no estar obligado a hacer las comprobaciones para asegurarnos de tener siempre el menor SF, los dispositivos traen un sistema de velocidad de datos adaptativa denominado ADR[6]. Suele ser conveniente tener habilitado este sistema en los dispositivos, siempre y cuando estos lo permitan. Es más, por encima de SF10 es obligatorio tenerlo activado. Y, para estar seguro de cumplir con los límites, es conveniente hacer uso de aplicaciones que ofrecen el cálculo de los límites de *airtime* para los diferentes SF y tamaño de los datos.

3.1.4. Clases A, B, C

Las clases de LoRaWan existen para ahorrar energía en los dispositivos. Esto lo consiguen al definir el tiempo en el que están a la escucha de mensajes del AP hacia el dispositivo en cuestión. Los dispositivos de la clase A y B, normalmente, son alimentados con baterías, por lo cual, intentan controlar al máximo el consumo. La clase A es la más eficiente porque después de haber enviado un mensaje, sólo está a la escucha durante dos intervalos de tiempo muy pequeños para recibir la respuesta desde el AP. El resto del tiempo están en modo ahorro de energía, y no escuchando activamente. La clase B tiene configurado intervalos fijos, que los sincroniza con el AP.

Los dispositivos de la clase C suelen utilizar una conexión directa a la corriente, debido a esto suelen estar todo el tiempo a la escucha de los mensajes, y pueden actuar de manera rápida si fuera necesario [44].

Normalmente, los dispositivos llegarán con una clase configurada, pero algunos tienen

[6]**ADR**: (Adaptive Data Rate) es un mechanismo para optimizar la transmision de datos, el *airtime* y el consumo energético. [43]

la opción de elegirla manualmente. Si no se requiere que el dispositivo responda en tiempo real a los mensajes *downlink*, es preferible utilizar la clase más alta (clase A).

El motivo por el que gestionar el tiempo de *downlink* activamente y estar dormido el resto de tiempo es debido a que LoRaWan está destinado mayoritariamente para subida de datos (*uplink*) y no para su recepción, ahorrando así más energía.

3.1.5. Autentificación

Para identificar el dispositivo dentro de la red utiliza el DevAddr[7]. Este está formado por el NwkAddr[8] y el NwkID[9]. Dependiendo del protocolo pueden estar predefinidos y guardados en el dispositivo para una red concreta, que es el caso de los dispositivos que utilizan ABP[10]. También puede ser obtenida dinámicamente a través de un procedimiento llamado “activation” al entrar en la red, siendo este es el caso para OTAA[11].

Siempre es recomendable utilizar OTAA, en vez de ABP, debido a que en este último los datos son fijos lo que puede resultar en problemas de seguridad. Y, si se cambian los frame counters, puede causar que los mensajes no sean aceptados por el gateway [45].

Después de probar ambos sistemas, hemos decidido que OTAA es la solución que más se adecúa a nuestras necesidades. Por lo cual, en este prototipo todos los sensores están configurados para utilizar este protocolo.

3.1.6. Ejemplo Milesight AM307

Para dar un ejemplo sobre los temas tratados con los sensores instalados, vamos a suponer el caso Milesight AM307:

Añadirlo a TTS

Dentro de TTS, tenemos la posibilidad de crear aplicaciones a las que podemos asignar dispositivos. Todos los dispositivos se tienen que dar de alta con un identificador único llamado EUI[12].

Para dar de alta un dispositivo, podemos comprobar si ya existe el modelo en el repositorio de dispositivos de TTS. Esto tiene la conveniencia de que ya viene con el PF definido y solo tenemos que insertar los datos que han venido con el dispositivo y que identifican al mismo (AppEUI, DevEUI, AppKey).

Así, ya tenemos dado de alta el dispositivo. Si los datos son correctos, y estamos al alcance de un gateway, ya debería de enviar datos. Sino, podemos probar cambiar el SF

[7] **DevAddr**: es un identificador de 32 bit que identifica el dispositivo en la red. Y, es utilizado para toda la comunicación en dicha red.[45]

[8] **NwkAddr**: es la dirección del dispositivo dentro de la red.[45]

[9] **NwkID**: es el identificador de la red. [45]

[10] **ABP**: (Activation By Personalization) tiene todos los identificadores, como por ejemplo el DevAddr fijos y codificados dentro del hardware del dispositivo y no los cambia durante la vida del producto. [45]

[11] **OTAA**: (Over-The-Air Activation) tiene que hacer un procedimiento de acceso a la red, en el cual, el dispositivo será asignado un DevAddr dinámico. [45]

[12] **EUI**: (también conocido como DevEUI de “End device identifier”) es un identificador de 64 bit, que identifica al dispositivo a nivel global. [45]

a uno superior y cambiar la periodicidad del envío de los datos para cumplir normas de *airtime*.

Calcular airtime

Los dispositivos instalados en la ULL están configurados para utilizar la frecuencia "Europe 863-870 MHz (SF9 for RX2)" y para utilizar como máximo SF9 con ADR activado. Mirando los datos que envían los sensores, podemos observar que, por el ADR, suelen trabajar en SF7. Si comprobamos los valores que ofrece la imagen, obtenidos a través de una calculadora de *airtime*[13], podremos observar que este sensor en SF7 podría enviar su payload aproximadamente 10 veces por hora. Debido a que están configuradas para enviarlos cada 20 minutos (o 10 minutos dependiendo del sensor), quedan dentro de los límites establecidos por TTN.

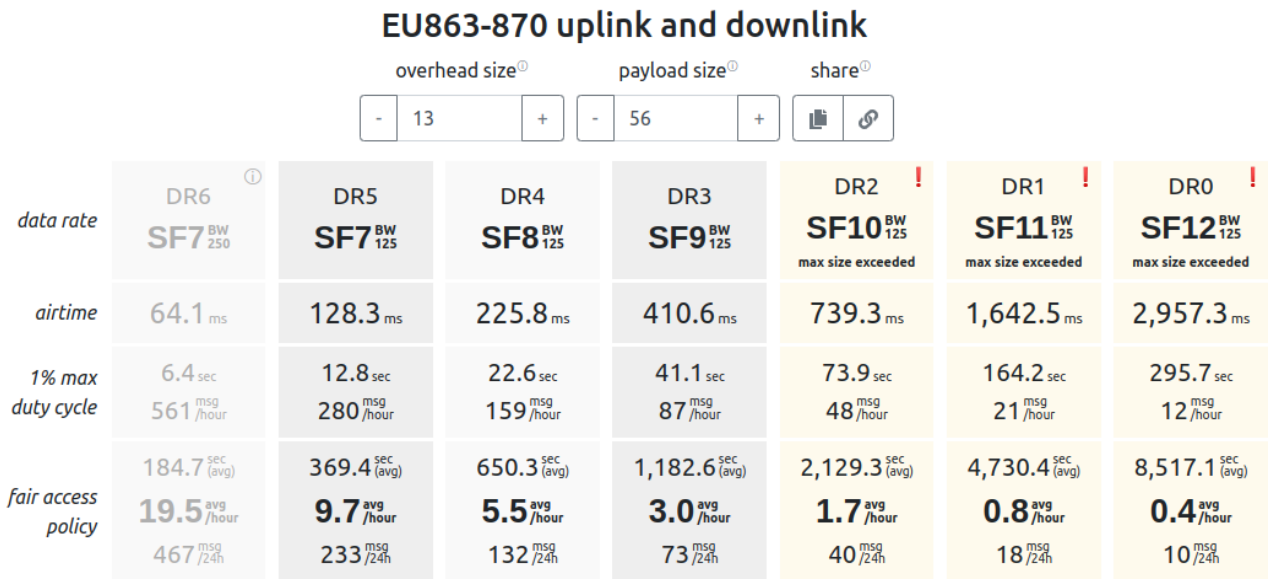


Figura 3.3: Calcular airtime para los diferentes spreading factors [46].

Payload Formatter

Como ya hemos comentado, los datos serán enviados en bytes. Es por esto que en el servidor de la aplicación se suelen incluir formateadores o "Payload Formatters" (PF) [47], tanto para el *uplink* como para el *downlink*. Esto también es el caso de TTS. Por lo tanto, el Backend recibe los datos ya transformados en un formato más legible, normalmente en JSON[14].

Para entender mejor cómo funcionan los PF de la aplicación, hemos incluido el ejemplo del PF para este dispositivo[15]

Los bytes recibidos pueden ser por ejemplo:

0367E400 04686F 050000 06CB02 077DE801 087D3A00 09736425 017564

[13] **calculadora de airtime**: Se trata de una calculadora que puede calcular el tiempo estimado de airtime después de que definamos el tamaño del payload del dispositivo. [46]

[14] **JSON**: (JavaScript Object Notation) es un formato de texto sencillo para el intercambio de datos.

[15] **Payload Formatter de Milesight AM307**: https://github.com/alu0101016733/TFG-IoT-alu0101016733/blob/master/payloadFormatter/milesightAM307.js

Y el PF los traduce a:

Después		Antes		
Que Sensor	Valor	Código	Tipo	Valor
Temperature	22.8	03	67	E400
Humidity	55.5	04	68	6F
PIR	0	05	00	00
LightLevel	2	06	CB	02
CO2	488	07	7D	E801
TVOC	58	08	7D	3A00
BatteryPercentage	100	09	73	6425
Pressure	957.2	01	75	64

Tabla 3.1: Ejemplo de la transformación llevado a cabo por el "payload formatter".

Más información en la documentación del sensor Milesight AM307 [48]

Normalmente, podemos obtener estos PF de la documentación de los fabricantes. Estos están escritos en JavaScript, lo cual nos permite cambiarlo para mejorar la claridad de comunicación con el backend. Por ejemplo, en esta implementación estamos utilizando el nombre de los sensores para identificarlos dentro de la BBDD.

3.1.7. Acceso a TTS

Para poder enviar los datos recibidos y formateados a una aplicación privada, TTS ofrece varias maneras, siendo las más interesantes los WebHooks y MQTT. Los WebHooks son una buena opción en caso de que la aplicación final tenga una IP fija, y pueda recibir datos desde internet. En otro caso, como es el de este proyecto, es mejor utilizar el servicio MQTT. Este lo hemos implementado como un microservicio del backend de la aplicación que actúa como el cliente de MQTT y que se subscribe al servidor de TTS. Los datos de autentificación del mismo los podemos obtener desde el apartado correspondiente (Integrations ->MQTT) de la aplicación creada en TTS.

3.1.8. Gateways

Normalmente, podemos hacer uso de los gateways dados de alta en TTN y no configurar nuestro propio gateway. Pero, puede ser interesante en caso de que no exista ninguno cerca. Esto lo podemos comprobar con antelación con el mapa de recubrimiento[7]. En nuestro caso, para poder tener más control, tener acceso a los logs y poder hacer pruebas más efectivas, resultó conveniente tener nuestro propio gateway. Es por eso, que también hemos dado de alta el gateway (RAK7258) como un gateway de TTS.

3.2. Backend

El backend de una aplicación suele ser el punto a través del cual toda la comunicación está dirigida y gestionada. Debido a esto, necesitamos un punto de partida estable. Es por ello, que después de hacer algunas pruebas de implementación en Python, hemos decidido mover todo al entorno de NodeJS y construir el backend sobre el framework NestJS. NestJS

trae un cliente para la creación del proyecto que nos ayuda en crear el proyecto principal y añadir recursos al proyecto creado y establecer las conexiones necesarias para que este recurso se integre directamente en la aplicación. Ya que trabaja sobre NodeJS, tenemos en la raíz del backend[16] el `package.json` con todas las librerías necesarias para el proyecto y el `tsconfig.json` para tener control sobre el comportamiento de TypeScript.

Dentro de `src` tenemos el `main.ts`[17] que es el fichero de entrada al proyecto, aquí definimos donde empieza (normalmente `app.module.ts`), que puerto está abierto a la escucha y algunos otros servicios que necesitan acceso a todo el proyecto, como por ejemplo Swagger UI[49].

En `app.module.ts`[18] ya define todos los import necesarios de los recursos a partir de la raíz del proyecto, ya que recursos anidados se integran a partir de su módulo padre.

Para hacerse un idea del proyecto, dentro de `src` tiene los siguientes carpetas:

- **`entity`**: Todo relacionado con la base de datos
- **`restApi`**: Todos las rutas a los recursos de la API
- **`subservice`**: Subservicios o microservicios no relacionados con NestJS
- **`utilities`**: Funciones que no se integran en NestJS
- **`ws`**: Contiene todos los WS

3.2.1. Base de datos

Para la integración de la base de datos dentro de NestJS tenemos múltiples opciones. El método básico es simplemente cargar el paquete de NodeJS para la base de datos que queremos. Pero también, ofrece la posibilidad de implementarlo con un ORM[19], más específico TypeORM, ya que está disponible en TypeScript. Al utilizar TypeORM tenemos un nivel más de abstracción que permite definir todas las tablas y consultas dentro de TypeORM, en vez de hacerlo para solo un único gestor de base de datos. Por lo cual, si fuera necesario, permite cambiar de manera muy fácil la base de datos integrada. Además, incluye un sistema de comprobación de tablas, que permite modificarlas o crearlas sin la necesidad de ejecutarlo en el gestor oportuno ya que TypeORM se encarga de crear y modificar las tablas necesarias en el momento de arranque. Eso sí, puede haber casos donde no se puede aplicar los cambios a la BBDD. En este caso tendremos que borrar la BBDD para que typeORM pueda generarla desde cero. Pero esto solo suele pasar en caso de quitar o modificar relaciones entre tablas.

Con respecto a dónde guardar la definición de las tablas, tenemos dos posibles aproximaciones. La implementada por defecto al utilizar el generador de recursos de NestJS[20]

[16] **Raíz del backend**: https://github.com/alu0101016733/TFG-IoT-alu0101016733/tree/master/backend

[17] https://github.com/alu0101016733/TFG-IoT-alu0101016733/blob/master/backend/src/main.ts

[18] https://github.com/alu0101016733/TFG-IoT-alu0101016733/blob/master/backend/src/app.module.ts

[19] **ORM**: (Object Relational Mapper) es una técnica de abstracción utilizada para poder implementar consultas en un lenguaje de programación. Estas consultas serán traducidos al lenguaje de la BBDD.

[20] **Generador de recursos de NestJS**: es una herramienta interactiva que ofrece NestJS para crear recursos y automáticamente enlazar a los ficheros correspondientes para que sea activa

es que cada tabla viene definido dentro del mismo recurso, junto a los dto[21]. Y, aunque en algunos casos estamos utilizando los dto, hemos optado por la manera de definir todas las entidades en una carpeta separada (`entity`). Esto permite mejorar la legibilidad en el proyecto, ya que además de las entidades en sí, había que definir algunos vistas y suscriptores que conectan las tablas, por lo cual es mejor tenerlos todos en un sitio. Otra razón por la que hemos optado por este método ha sido que debería permitir diferentes métodos de acceso a los recursos.

La representación de las tablas se hace a través de clases. Haciendo uso del patrón de diseño de software decorador (@Entity, @Column, @PrimaryGeneratedColumn, @OneToMany, etc) que ofrece TypeORM y NestJS. Este patrón permite a NestJS y TypeORM comunicar a las librerías integradas la estructura de datos que tienen.

Para dar una idea básica sobre los datos guardados y las relaciones entre ellos tenemos el siguiente Modelo ER creado con phpMyAdmin[22]:

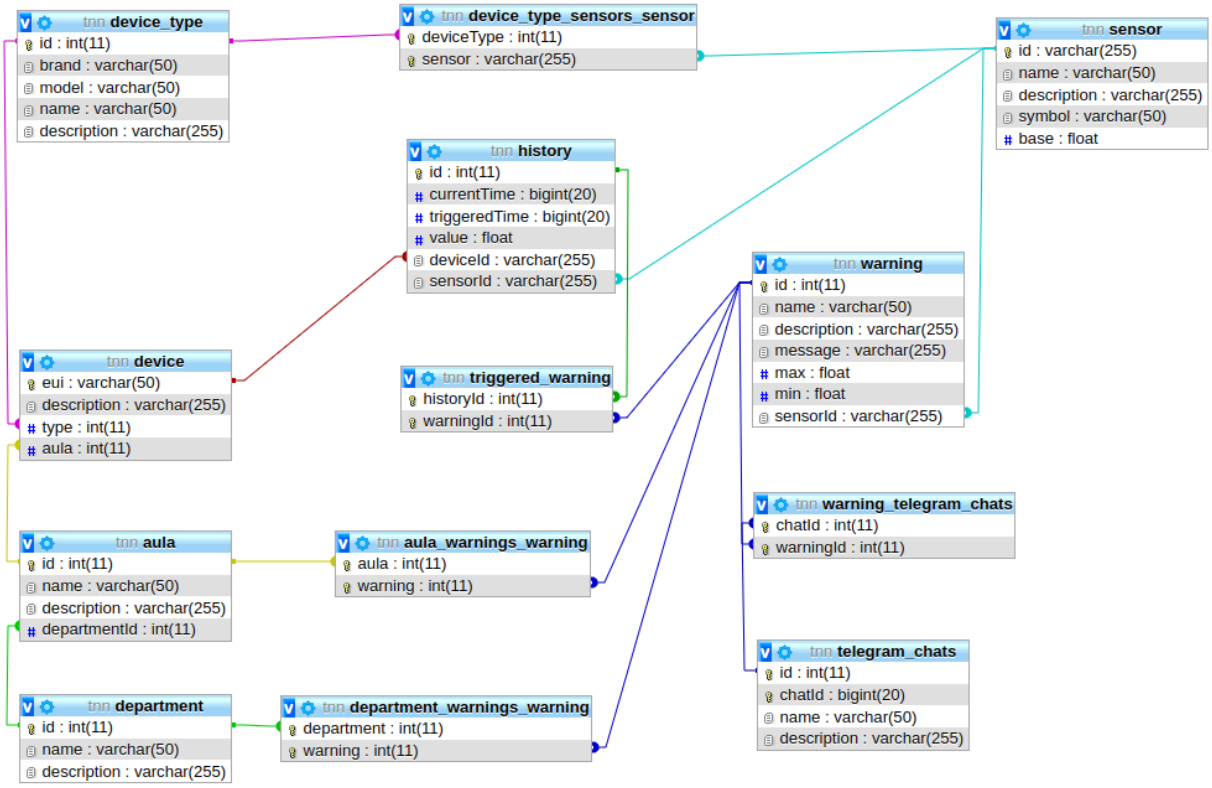


Figura 3.4: Modelo entidad relación de la base de datos.

A continuación expondremos algunos detalles sobre las tablas más importantes.

- **device**: representa un dispositivo físico instalado. Se puede observar que utiliza como clave primaria el EUI del dispositivo para identificarlo de la misma manera de TTS: este dispositivo pertenece a un tipo y está instalado en un aula concreta.

[21] **dto**: (Data Transfer Object) especifica el formato u objeto JSON que espera recibir en el body de la llamada a este recurso.

[22] **phpMyAdmin**: es una herramienta que permite la administración de BBDD tipo MySQL sobre la web[50].

- **sensor**: contiene la información relativa de un sensor.
- **device_type**: como dice el nombre, son tipos de dispositivos. Muchos de los aparatos de IoT del mercado ofrecen múltiples sensores integrados. Esta tabla permite guardar información sobre los dispositivos y de los sensores que incluyen.
- **history**: esta tabla guarda toda la información de los datos obtenidos de los distintos dispositivos.
- **warning**: define los valores máximos y mínimos a partir de los cuales es necesario avisar a los usuarios.
- **triggered_warning**: guarda la información de las advertencias activadas en su momento.
- **telegram_chats**: permite guardar la información de un chat concreto de Telegram, que después puede subscribir a diferentes alertas si fuera necesario.
- **aula** y **department**: representan agrupaciones en instalaciones físicas. Un punto interesante podría ser definir ciertas alarmas solo en algunas aulas o departamentos concretos, por lo cual tenemos una conexión de los mismos con la tabla warning.

Vistas

Para facilitar al usuario un acceso eficiente a los datos, se han creado algunos vistas que incluyen la información necesaria para las consultas, uniendo las partes en el backend. Tienen la misma lógica que las tablas normales de TypeORM con la diferencia que el decorador utilizado es @ViewEntity. La consulta que crea la view es pasada como argumento a este decorator.

Suscriptores

En TypeORM tenemos dos posibilidades de ejecutar algún código en un momento concreto de la comunicación con la base de datos[51].

- **Entity Listeners**: se definen dentro de la @Entity y por encima de la función que se quiere ejecutar. Esto está bien si trabaja solo con la misma tabla. Por ejemplo, en caso de que queramos hacer algo antes de insertar añadimos @BeforeInsert a la tabla.
- **Subscriber**: estos se definen fuera de la @Entity con el decorador @EventSubscriber y permiten escuchar a todas las entidades o solo una en concreto. Hemos optado por esta metodología porque es más limpio tener todos los suscriptores en un único sitio.

En la aplicación estamos utilizando los Subscriptores para comprobar si hay que lanzar una alerta o no, e informar a los usuarios de los cambios a través de los WS definidos para ello.

3.2.2. REST API

Como nombramos en el apartado de Base de datos, NestJS dispone de un método para crear los ficheros necesarios para la representación de un recurso dentro de nuestra RestAPI. Esto es de gran ayuda porque deja el proyecto más ordenado y permite a otros desarrolladores acostumbrados a trabajar con NestJS, sin mayor esfuerzo, seguir con el proyecto.

Esto significa, que cada recurso accesible desde la API es representado con una carpeta dentro de la carpeta `restApi` del proyecto. También nos permite crear estructuras dependientes. Así, por ejemplo, podemos decir que queremos acceder a todas las aulas dentro de un departamento. En este caso, el departamento es el primer punto de partida (/department). A partir de ahí, podemos definir aulas dentro de departamentos y obligar al usuario a que proporciona el identificador del departamento antes de poder acceder a las aulas (/department/{departmentId}/aula).

Con respecto a los archivos. Dentro de un recurso que representa la RestApi, siempre vamos a tener 3 ficheros:

- **Module**: (decorador @Module) contiene la metadata que es requerido por NestJS para saber en qué orden debe crear la aplicación. Aquí necesitamos definir todas las dependencias que tiene este módulo.
- **Controlador**: (decorador @Controller) es el responsable de manejar las consultas y define la ruta en la que está disponible este dato. Dentro de esta clase podemos definir las acciones que permite este recurso (@Post, @Get, @Delete, @Patch). Si queremos que sea visible al usuario dentro de Swagger, tenemos que añadir @ApiParam definiendo el nombre y tipo. También veremos que es dependiente de Service de este recurso.
- **Service**: (@Injectable) contiene todas las funciones necesarias para hacer las consultas y es el único fichero de los tres que puede acceder a los recursos de otros ficheros mediante el decorador @InjectRespository.

Pero, aunque se esta utilizando la misma estructura, también estamos haciendo uso de la librería Nestjsx Crud[52]. Ésta ofrece una manera simplificada de crear puntos de acceso a los recursos. Además, incluye otras funcionalidades ya implementadas como, por ejemplo, la elección de columnas, filtros, etc[53].

La implementación se hace a través del decorador @Crud de esta librería dentro del controlador del recurso. Y permite especificar qué solicitudes de HTTP están disponibles, qué filtros se pueden añadir, qué campos pertenecen a las variables de la URL y si el usuario puede obtener información de otra tabla vinculada[54]. Pero debido a que implementa muchas acciones, limita un poco el uso de los mismos, sobre todo tiene problemas con implementaciones hechas a partir del decorador @ManyToMany de TypeORM. Aun así, debido a que NestJS persigue la misma lógica, podemos emplear la librería en los casos posibles y hacer la implementación manual en otros casos.

Documentación (Swagger UI)

Para poder comprobar el funcionamiento de la API y proporcionar al usuario la documentación, se ha utilizado la librería Swagger UI. La ventaja de esta librería es que también tiene un módulo para NestJS y que es compatible con la librería NestJSX-CRUD.

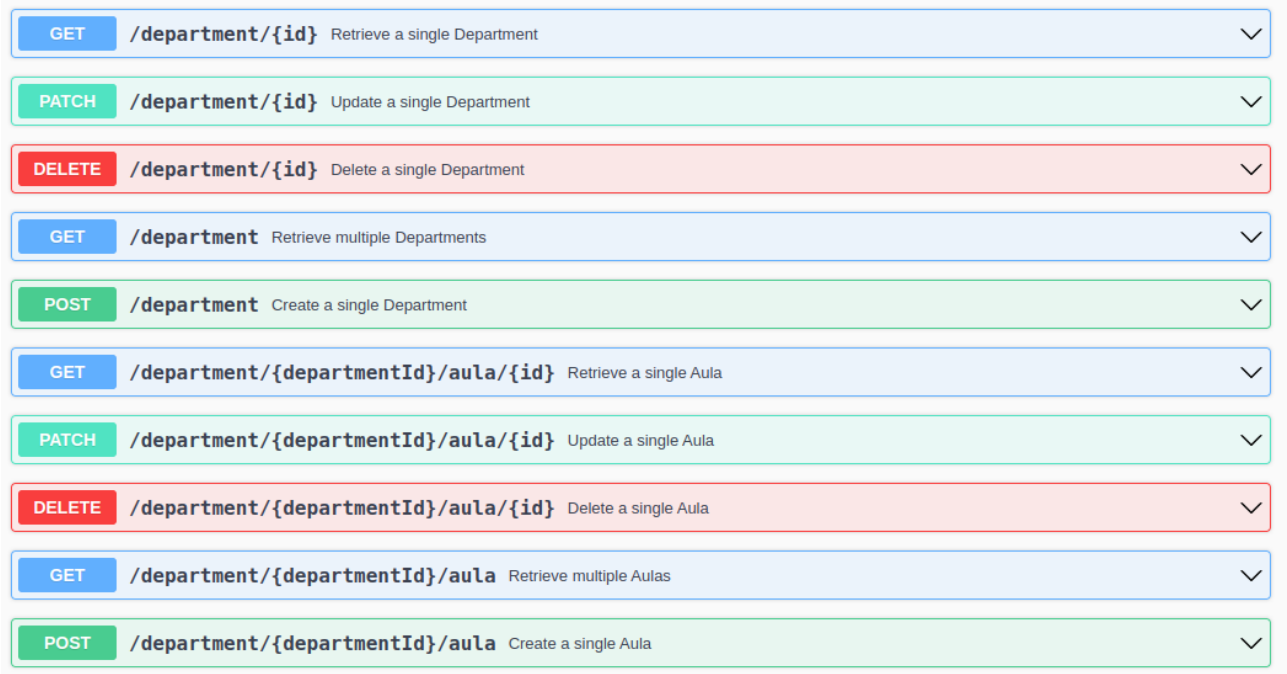


Figura 3.5: Primeros entradas del documentador Swagger UI de la API.

Añadirlo al proyecto es tan simple como añadir el módulo de swagger en el `main.ts` del proyecto. A partir de ahí, ya empieza a funcionar. Pero, para que pueda saber los datos requeridos por cada recurso, tenemos que añadir a los modelos de tablas el decorator @ApiProperty. Esto nos permite presentar al usuario el formato de dato que tienen las consultas y los datos requeridos en caso de crear o modificar elementos.

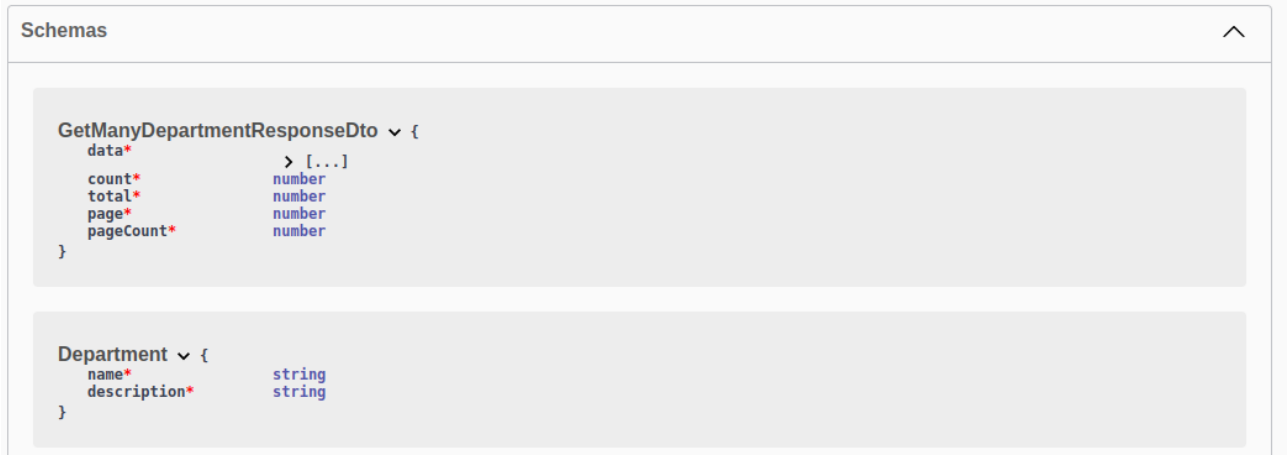


Figura 3.6: Ejemplo de esquemas definidos en Swagger UI.

3.2.3. WebSocket

Como ya se comentó, algunos mensajes se hacen a través de WS. La ventaja es que NestJS ofrece librerías internas para facilitar el despliegue de los mismos, pero hay que

tener en cuenta la librería utilizada. Esto es debido a que con algunas librerías no se permiten utilizar los WS integrados en los navegadores sino que necesitan herramientas externas. Por esto es conveniente emplear alguna librería como socket.io[55], que sí permite utilizar la solución integrada.

Los WS tienen la misma lógica interna que la CRUD API, pero usan otras funciones y decoradores para permitir el control de los WS. Uno de los decoradores más importantes es el de @SubsribeMessage, que permite definir las funciones a ejecutar dependiendo del evento que viene. En esta implementación, por ejemplo, el usuario se conecta al WS para recibir los datos de los sensores en directo, pero antes de que sea añadido, hace falta que envíe un mensaje con el evento "subscribe".

3.2.4. Cliente MQTT

Como hemos nombrado anteriormente, estamos haciendo uso del servidor MQTT de TTS para recibir los datos desde el backend. Por lo cual, dentro del backend hemos creado un microservicio como cliente MQTT. Esta decisión ha sido tomada debido a que no forma necesariamente parte de la API. Pero, no tenía mucho sentido crear otro proyecto aparte para este servicio, ya que en este momento solamente se encarga de recibir los datos de una única aplicación de TTS y los inserta a través de la API en la BBDD. Aun así, en el supuesto caso de que queramos manejar varias aplicaciones sería conveniente separarlo en otro proyecto. Ya que puede crecer rápidamente y limitar la manejabilidad del mismo. De esta forma, la implementación como microservicio hace que actúe como un servicio independiente del resto de los ficheros. Por lo cual se podría mover fácilmente a otro proyecto. Es por esto que este servicio utiliza la misma comunicación con la Aplicación como si fuera independiente (En lugar de utilizar los @Injectable de NestJS hacemos uso de la librería @nestjs/axios[56] para enviarlo como solicitud HTTP).

3.2.5. Bot de Telegram

El bot de Telegram también está incluido como microservicio por razones parecidas a los del cliente MQTT. Éste se suscribe a los WS de alertas y en caso de que reciba alguna, hace las consultas necesarias para obtener la información necesaria desde la API para informar a los usuarios suscritos a esta alerta, de la misma. Para la comunicación con el bot se hace uso de la librería node-telegram-bot-api[57] y un bot que hemos creado desde Telegram con nombre “alertTelBotTest334bot” que es el encargado de enviar los mensajes a los chats definidos en la BBDD. Es importante que el bot esté incluido en el chat. Esto debido a que Telegram, por razones obvias, no permite que bots se añadan de manera automática a grupos o que envíen mensajes a chats donde no están incluidos como miembros.

3.3. Frontend

Como hemos comentado en el estudio previo, la parte del Frontend[23] se llevó a cabo con el framework React y algunas librerías adicionales para el diseño y comunicación entre componentes.

[23] **raiz frontend**: https://github.com/alu0101016733/TFG-IoT-alu0101016733/tree/master/frontend

El diseño de la aplicación se ha hecho utilizando algunos componentes de Material-UI[58] junto a la modificación o adición de los estilos requeridos para mejorar la apariencia. Hay que tener en cuenta, que estamos empleando la versión 18+ de React, por lo cual, solo podemos hacer uso de las funcionalidades compatibles con esta versión de React. Así, por ejemplo, si queremos añadir el tema claro/oscuro, tendríamos que cambiar todo el estilo como está hecho hasta este momento e implementarlo de la manera alternativa que ofrece Material-UI. Es por eso, que para la implementación de los temas claro y oscuro se ha programado el componente ThemeProvider, el cual será explicada más adelante.

En el *layout* principal de la página se ha tenido en cuenta la importancia de las alertas. Por lo cual, el usuario siempre tiene la cantidad de alertas o fallos de configuración visibles en la parte derecha superior de la pantalla.

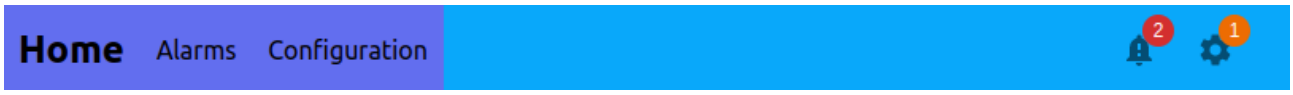


Figura 3.7: Barra principal del frontend con alerta activas.

Para conseguir que estos datos estén disponibles en todos los componentes, pero, que solo se rendericen los que realmente tienen que trabajar con ellos, hemos utilizado la librería recoil[59] (otra opción hubieran sido los Contextos de React[60]). Es decir, tanto las alertas como los errores de configuración pueden ser mostrados u ocultados en cualquier momento en la parte derecha de cualquiera de las pantallas de la página web. Esta decisión de diseño ha sido tomada para facilitar y para tener siempre disponible la lectura de estas notificaciones. Además de poder tenerlo siempre visible, hemos implementado la opción de que el usuario pueda cambiar el tamaño de este componente, para así ofrecerle la posibilidad de leer mejor las alertas, en caso de que haya muchas o que los mensajes de estas sean muy extensos.

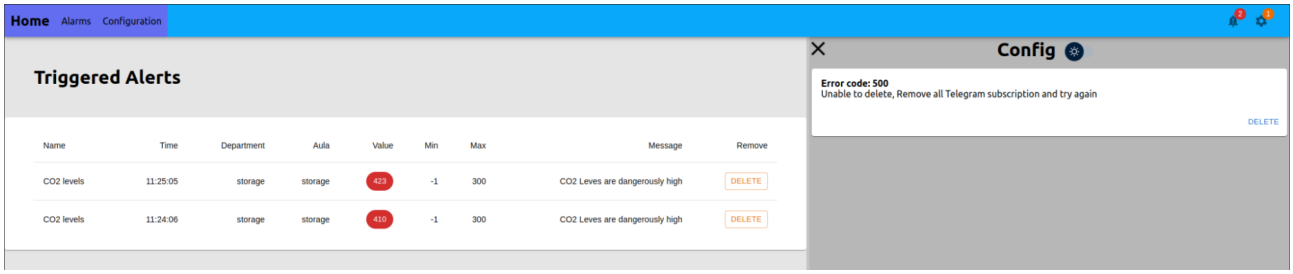


Figura 3.8: Parte derecha de la página extendida.

Aparte de esto, hemos dividido la página en tres grandes bloques que se explican a continuación. Cada página y las páginas que contienen, están siendo apuntadas por una URL y manejadas con la librería React Router[61].

3.3.1. Página de inicio

La primera página, “Home”, está enfocada en la visualización de datos. Para esta página hemos creado un componente que deja, al usuario final, dividir la pantalla en las secciones que considere necesario para visualizar los datos mediante gráficos. En cada subventana, el usuario tendrá la posibilidad de elegir los departamentos, aulas y

sensores que quiere visualizar. Para dar incluso más control sobre la apariencia de la página principal, el usuario puede cambiar el tamaño que tiene cada subventana.

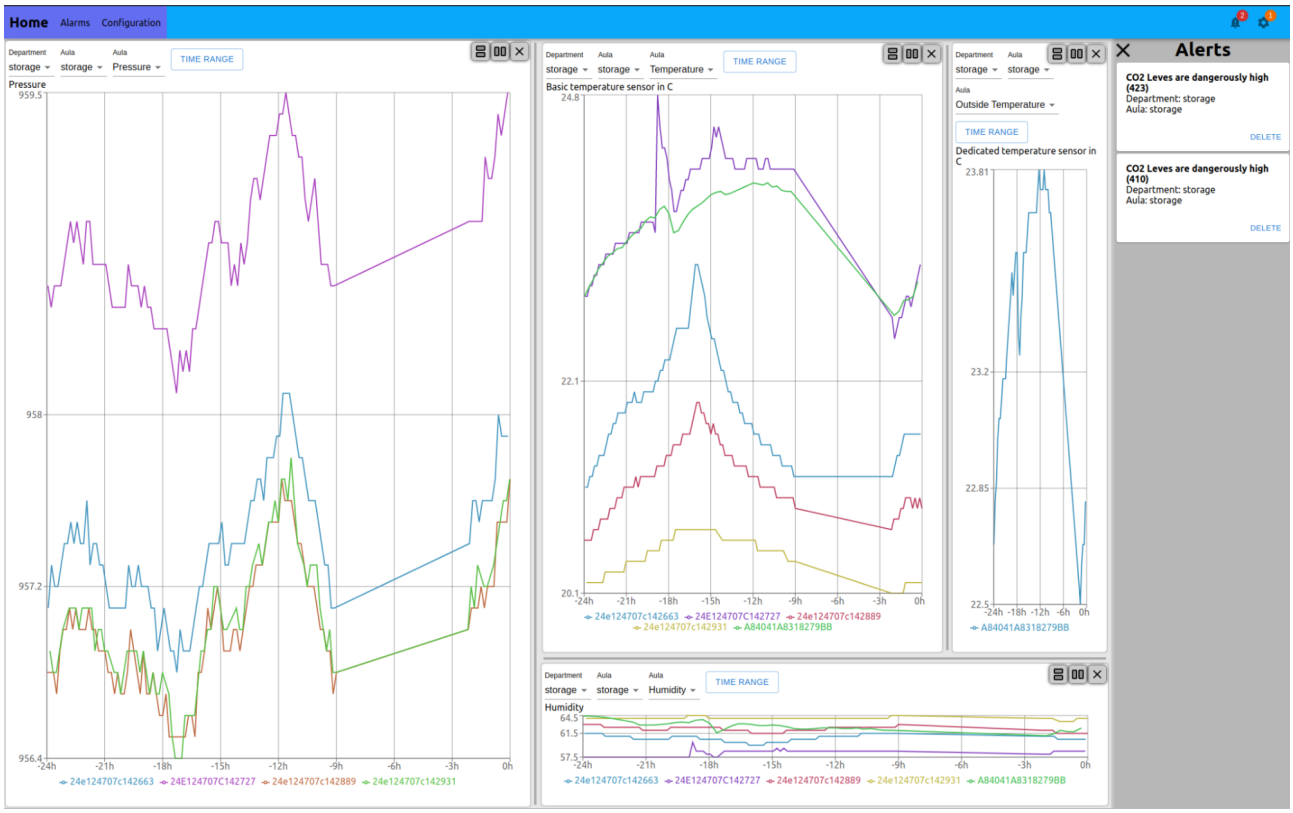


Figura 3.9: Página principal del frontend partida en subventanas.

Para facilitar la experiencia de usuario, se ha establecido que se guarde la información acerca de la configuración del diseño y de los sensores visualizados en cada momento en el *local storage*. En el caso de que el usuario cambia la selección el local storage será actualizado automáticamente. De esta manera conseguimos que el usuario no tenga que configurar la interfaz que quiere utilizar cada vez que accede a la página web de nuevo.

Al iniciar por primera vez la página, para que el usuario pueda ver las diferentes opciones que tiene y no presentarle el home vacío, se hacen automáticamente consultas a la API y se elige el primer sensor activo. Es decir, el sensor en el cual en algún momento haya habido datos históricos. Después, el usuario tiene la posibilidad de cambiar el sensor, eligiéndolo de las opciones que se presentan en la parte superior izquierda de cada gráfico.

En cuando abrimos la opción de “Time range” se nos presenta un modal[24] que pregunta por el modo a utilizar. Tenemos dos posibles opciones:

- **“Last Hours”**: es la opción con la que podemos definir el rango de tiempo que queremos ver. Así podemos, por ejemplo, decidir que este gráfico represente las últimas 12 horas de datos. Esta opción, debido a que son siempre las últimas horas, no quita la actualización automática en caso de que haya nuevos datos.

[24] **modal**: es una ventana emergente, normalmente un formulario o aviso, que se superpone al contenido normal de la página.

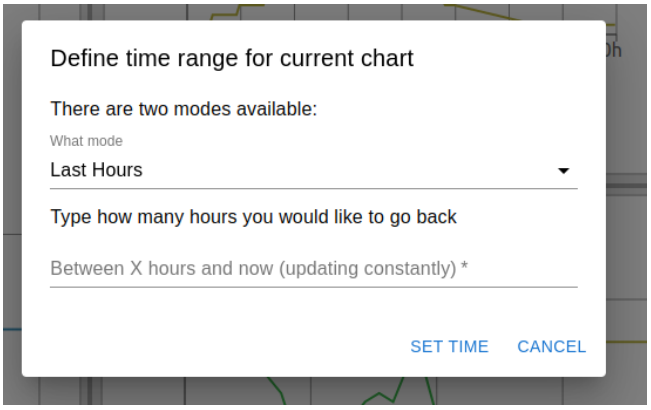


Figura 3.10: Modal para cambiar las últimas horas visualizadas en el gráfico.

- **"Define range"**: es una opción que nos permite elegir el rango de tiempo que queremos visualizar. Esta función puede ser muy útil cuando queremos comparar diferentes días o años. Podríamos, por ejemplo, partir la ventana principal en dos subventanas y elegir en una el mes pasado y en la otra el mes actual para visualizar las diferencias que tienen.

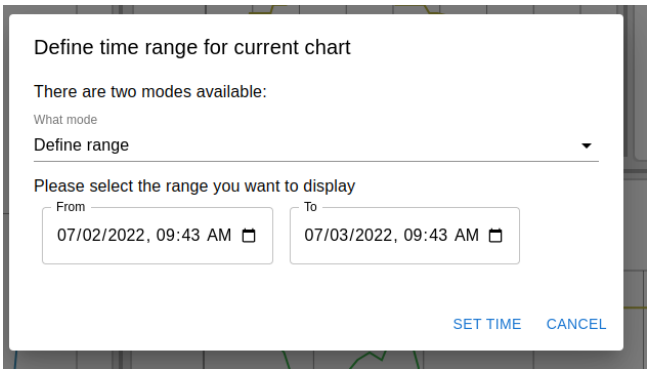


Figura 3.11: Modal para visualizar un rango fijo en el gráfico.

Cada componente gráfico genera su propia conexión a un websocket del backend y se suscribe solo a los valores que el usuario quiere visualizar en cada momento. Por lo cual, cada gráfica se actualizará de manera independiente y, exclusivamente, cuando el sensor al que está suscrito tenga nuevos valores.

3.3.2. Página de alertas

La página de alertas contiene un listado en forma de tabla de las alertas que fueron enviadas. Esta pantalla ha sido creada para poder visualizar información adicional, ya que también tenemos la información básica en el desplegable de la parte derecha. En ambas secciones, tanto en la pantalla de alarma, como en el desplegable de la derecha, se pueden eliminar alertas y gracias a utilizar la librería recoil, ambos listados se actualizan automáticamente.

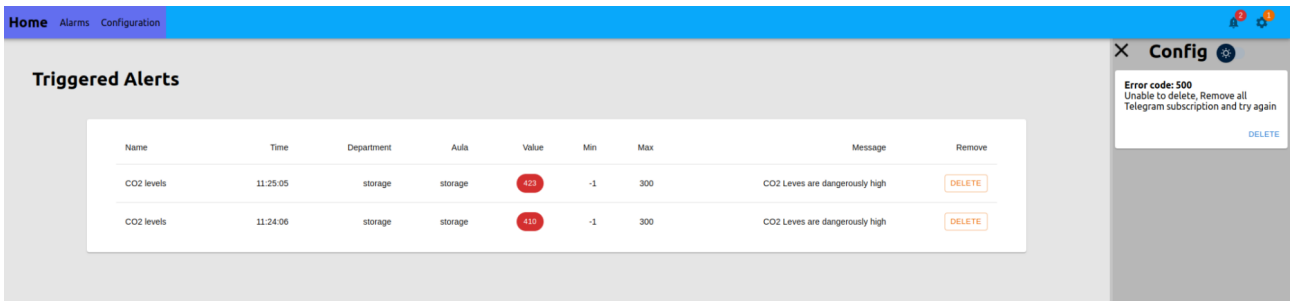


Figura 3.12: Página de alertas del frontend con dos alertas.

3.3.3. Página de configuración

Para facilitar la configuración al usuario final, tenemos la página de configuración. En esta página tenemos al lado izquierdo acceso a todas las posibles opciones. Por ejemplo, podemos configurar alertas en la ventana de Warnings. Entrando en esta página, podemos ver que nos presenta la lista de las alertas configuradas en la BBDD.

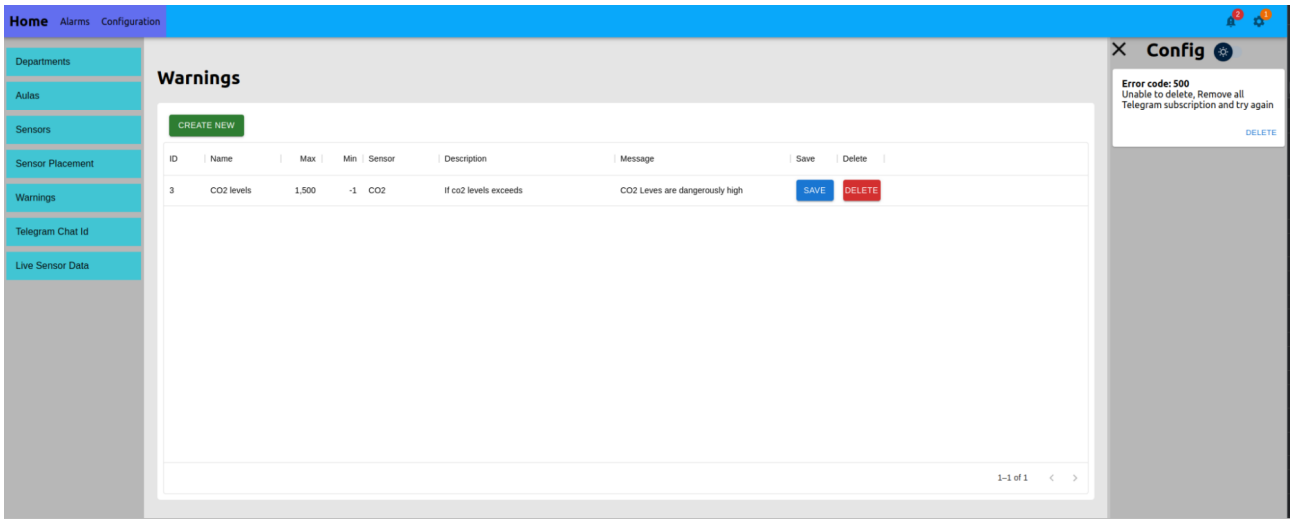


Figura 3.13: Página de configuración del frontend, representando la vista de alertas configuradas.

Pero esta lista no solo permite visualizar los valores, sino que además integra opciones de ordenar los resultados y borrarlos, en caso de que ya no hagan falta. También permite que el usuario edite los campos en tiempo real para actualizar la información. Una vez que el usuario guarda los resultados, estos serán guardados en la BBDD. En el caso de que haya habido algún problema con la actualización, se informa al usuario y se actualiza el valor para que vuelva a representar el contenido de la BBDD.

Figura 3.14: Página de configuración del frontend, representando edición de valores de alertas ya creadas.

Para crear nuevas entradas se abre un modal que permite al usuario rellenar la configuración necesaria y, en caso de que dependan de un valor definido, se podrá elegir desde un dropdown.

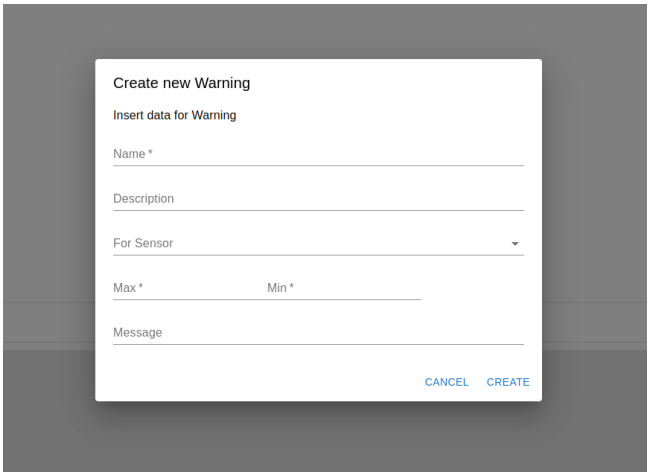


Figura 3.15: Página de configuración del frontend, modal de crear nuevas alertas.

Al momento de crear la nueva entrada, se reflejan los cambios en el listado o se informa al usuario del error ocurrido durante la inserción.

Si alguna entrada depende de otros valores, como puede ser el caso de configurar el bot de Telegram, se presenta un botón adicional en la lista.

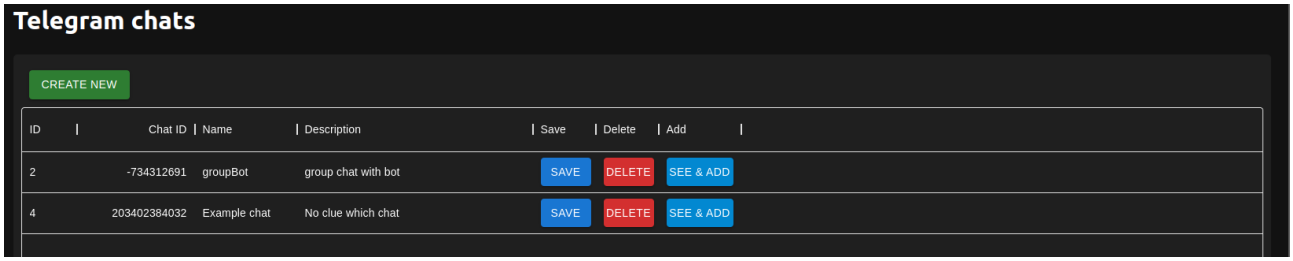


Figura 3.16: Página de configuración del frontend, Lista de chats configurados (estilo oscuro).

Esta lista solo presenta los chats de Telegram definidos. Pero, para tener control de las alertas, podemos utilizar el botón SEE & ADD para configurar las alarmas a las que este chat se suscribe.

Ya que solamente tiene un único valor definido, no se vio la necesidad de que el usuario tenga que guardar después de cada acción. Por lo cual, eligiendo una alerta desde el desplegable, se añade automáticamente la entrada a la BBDD y actualiza la lista de suscripciones. En esta lista tendremos la posibilidad de borrar las suscripciones para que no nos lleguen las alertas.

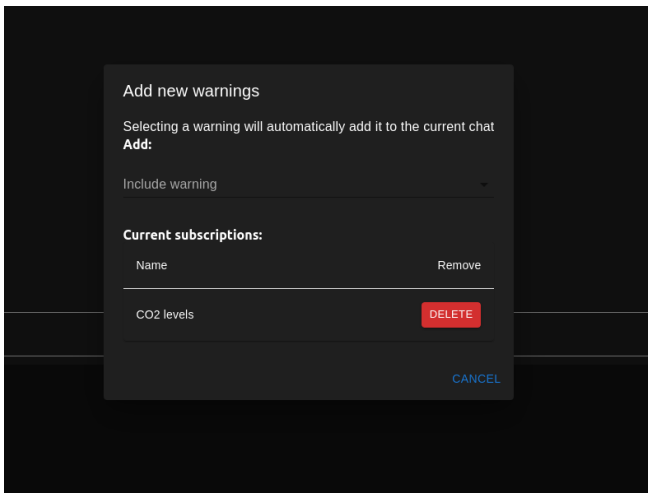


Figura 3.17: Página de configuración del frontend, Modal de suscripciones de Telegram (estilo osculo).

Para dar una idea de que hace cada página dentro de la configuración, vamos a resumirlo aquí:

- **"Departments"**: Aquí se pueden crear y modificar los diferentes departamentos.
- **"Aulas"**: Permite, una vez seleccionado un departamento, añadir y modificar aulas.
- **"Devices"**: Los dispositivos siempre suelen encontrarse en un departamento y una aula. Por lo tanto, para añadir y modificar dispositivos hay que seleccionar primero el aula en la cual se encuentran en este momento.
- **"Sensor placement"**: Nos permite cambiar el dispositivo del aula. Para esto, tenemos que elegir primero el aula de origen, y después, la nueva aula en la cual está ahora.
- **"Warnings"**: Permite la creación y modificación de alertas. Además, permite eliminar alertas, pero solo si no hay usuarios de Telegram suscritos a ellas.
- **"Telegram Chat Id"**: Permite añadir nuevos chats. El bot de Telegram comunicará, a través de estos chats, de las nuevas alertas que ocurran a los usuarios suscritos.
- **"Live Sensor Data"**: Es una lista donde se reflejan los últimos 50 datos que nos llegaron de los sensores.

3.3.4. Componentes a destacar

Como ya se ha mencionado, se han creado algunos componentes a mano para generar los comportamientos esperados. De la lista de los componentes creados nos gustaría señalar:

Divididor de pantalla (Split Screen)

Como ya hemos nombrado, tenemos un componente que puede partir la pantalla en varias más pequeñas[25]. Además, esta división puede ser tanto horizontal como vertical

[25] `https://github.com/alu0101016733/TFG-IoT-alu0101016733/blob/master/frontend/src/components/splitScreen/SplitScreen.tsx`

para aumentar la configurabilidad de la interfaz. Para no sobrecargar al usuario con la configuración, se ha creado una interfaz muy simple, donde el componente solo espera un ReactNode[26] , del cual va a partir. Es decir, este ReactNode va a ser la base de todas las ventanas del componente. Por lo cual, es recomendable que sea un componente configurable a través de las interfaz.

En este proyecto, SplitScreen está siendo utilizado para los gráficos y, debido a que los gráficos necesitan algún tiempo de ajustarse a un nuevo tamaño, se hace uso del React Hook useMemo[27] para los componentes que se añaden al SplitScreen. Esto permite, únicamente renderizarse en caso de que haya cambios en el *layout*. Con esto conseguimos que la interacción con el usuario sea más rápida. Pero, por contraparte, no refleja los cambios en los instantes en los que el usuario cambia el tamaño del gráfico, sino que se espera a que el usuario haya terminado de ajustar las diferentes ventanas.

Internamente, este componente funciona de manera recursiva y está estructurado como un árbol binario, por lo cual, solo se puede cambiar el tamaño entre dos ventanas. Pero, debido a que estas ventanas pueden tener otras dos dentro, de forma recursiva, en caso de que hubiera cambios se tenía que informar a ambas ventanas hijas. Para esto se ha implementado una lógica inspirada en los eventos de JavaScript[28].

El algoritmo desarrollado para transmitir los eventos entre el árbol binario generado consta de varios pasos: el primer paso es la fase de Bubbling, que se ejecuta primero en el componente que ha cambiado y sube hasta la raíz. Una vez llegado a esta, entra en la fase de Capturing, donde desciende desde la raíz hasta, y aquí está la principal diferencia con los eventos de JavaScript, todos los elementos hijos, informando a estos del cambio.

Gráficos

Al principio, se ha implementado con la librería react-chartjs-2[63] . Pero en el momento de crear la aplicación, descubrimos que esta librería no es del todo compatible con la versión de React utilizada, por esto se ha cambiado la aplicación a la librería recharts[64].

Para que el usuario pueda utilizar este componente para la visualización, hemos añadido la posibilidad de que controle automáticamente los colores de las líneas de los datos representados. Para este caso, hemos originado dos algoritmos diferentes. El primero, normalmente destinado a representar diferentes departamentos, divide el espacio de colores posibles en el número de departamentos que se quieren representar, además define un rango cercano para posibles colores de las aulas. De momento, lo empleamos para representar los diferentes sensores dentro de una aula en el tema claro.

[26] **ReactNode**: es un tipo que representa un elemento sin estado, inmutable y virtual del DOM.

[27] **useMemo**: es un Hook que solo ejecuta una función al principio, y solo invoca la misma en caso de que las dependencias hayan cambiado.

[28] **eventos de JavaScript**: cuando se ejecuta un evento se JavaScript, se consideran tres fases: Capture, Bubbling y Target. [62]

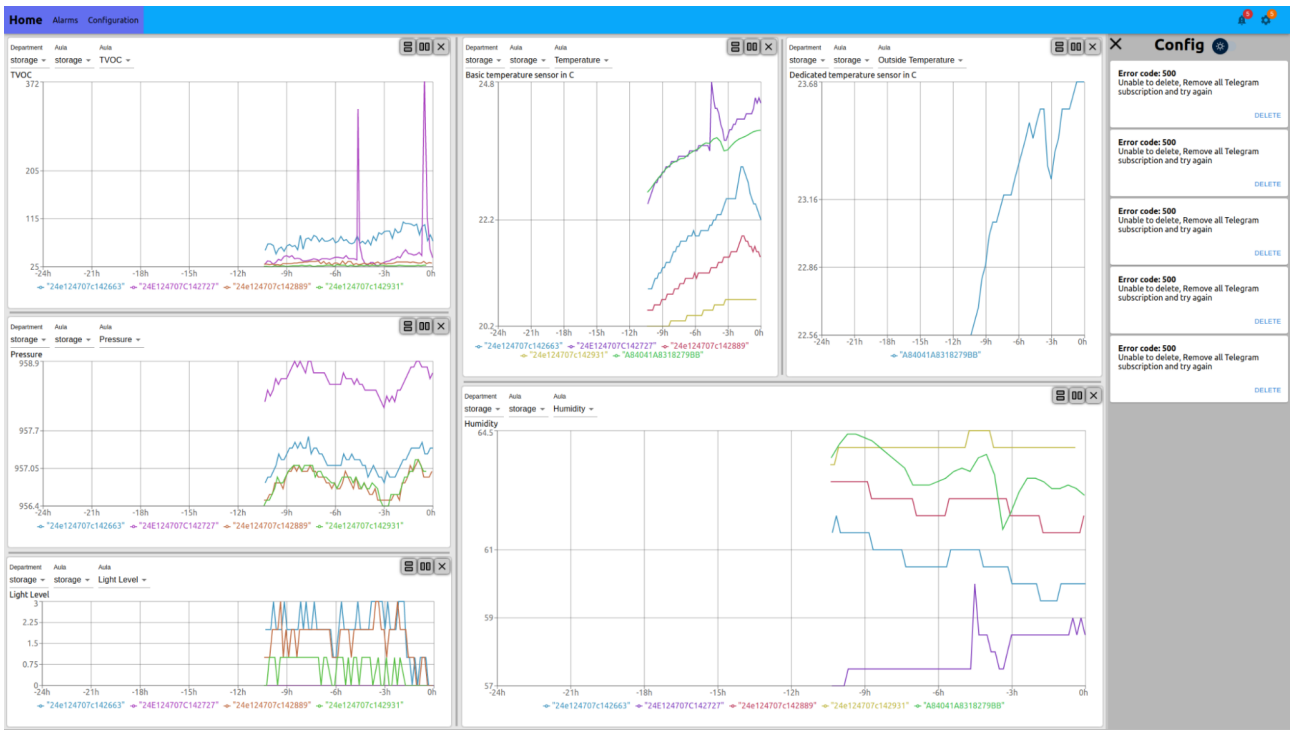


Figura 3.18: Página principal con el algoritmo de obtención de colores en grupos de departamentos (tema claro).

El segundo comienza con los colores bases y los rangos que se obtuvieron del primer algoritmo para elegir colores cercanos. Normalmente, está diseñado para agrupar las aulas dentro de los diferentes departamentos. Así, por ejemplo, las aulas dentro de un departamento pueden tener diferentes tonos de azul, o los valores cercanos al mismo rango obtenido en la agrupación de los departamentos.

Adicionalmente, al ser un componente versátil, podemos ajustar los datos representados para que este escoja un rango de colores, para las líneas de los gráficos, independientemente del tema elegido claro u oscuro. Ahora mismo, en la página principal, se utiliza el tema oscuro para representar los diferentes sensores en una aula.

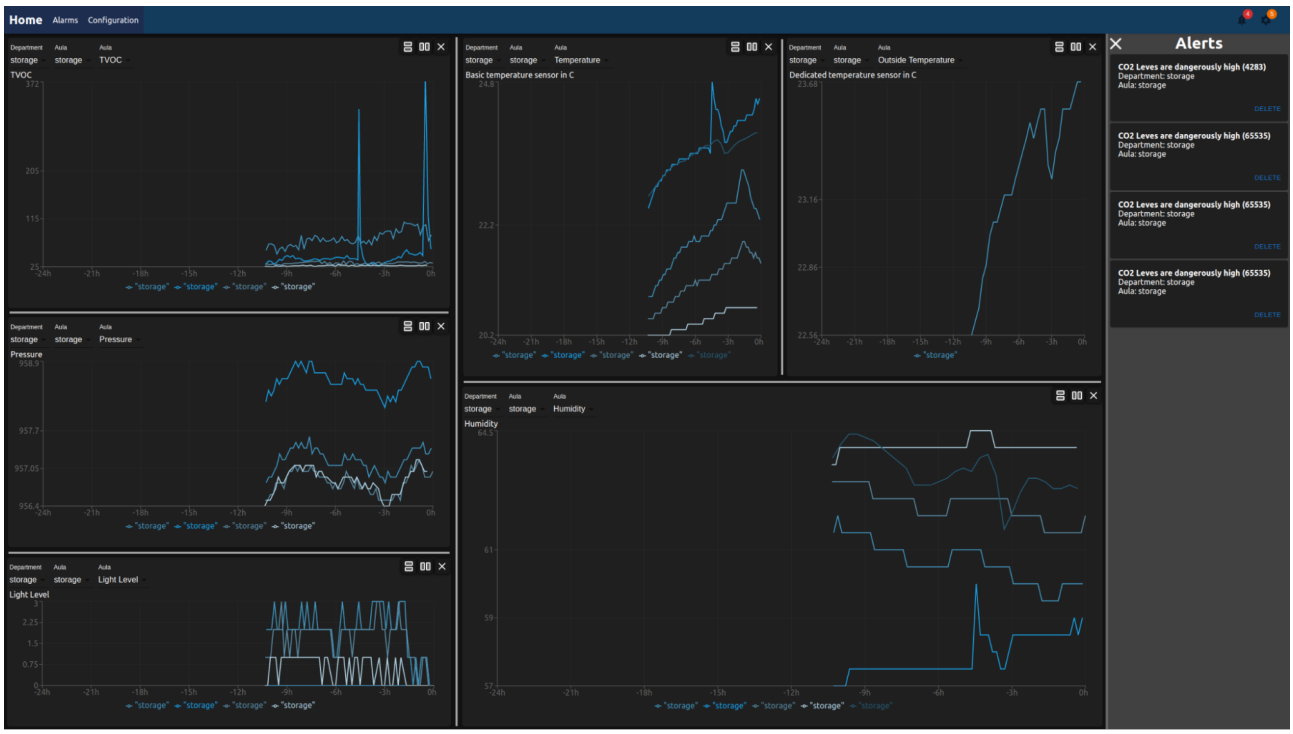


Figura 3.19: Página principal con el algoritmo de obtención de colores en grupos para aulas (tema oscuro).

Provedor de temas (ThemeProvider)

Aunque tenemos un proveedor de temas dentro de material UI, hemos decidido escribir nuestro propio Hook[29] para tener incluso más control sobre su comportamiento. Esta decisión se ha tomado debido a que estamos utilizando algunos componentes propios y, en un futuro, es muy probable que se implementen más componentes propios. Por lo cual, es conveniente tener control completo sobre el comportamiento del tema. Esto también permite, utilizando las funcionalidades de la librería @emotion/css[65], escribir el código css dentro de ficheros TypeScript y así tener acceso a las variables que configuran el tema.

De momento, solo están implementados dos temas diferentes: oscuro y claro.

Figura 3.20: Botón animado dentro de los ajustes para cambiar entre los temas.

3.4. Comunicación entre componentes

Ya que en el IoT es muy importante la comunicación entre los diferentes componentes dentro de una aplicación, vamos a entrar un poco en el modelo de comunicación que utiliza la aplicación.

[29] `https://github.com/alu0101016733/TFG-IoT-alu0101016733/blob/master/frontend/src/modules/ThemeProvider.tsx`

Para esto, seguiremos los diferentes pasos que debe pasar un paquete en el momento de que sea enviado desde un sensor.

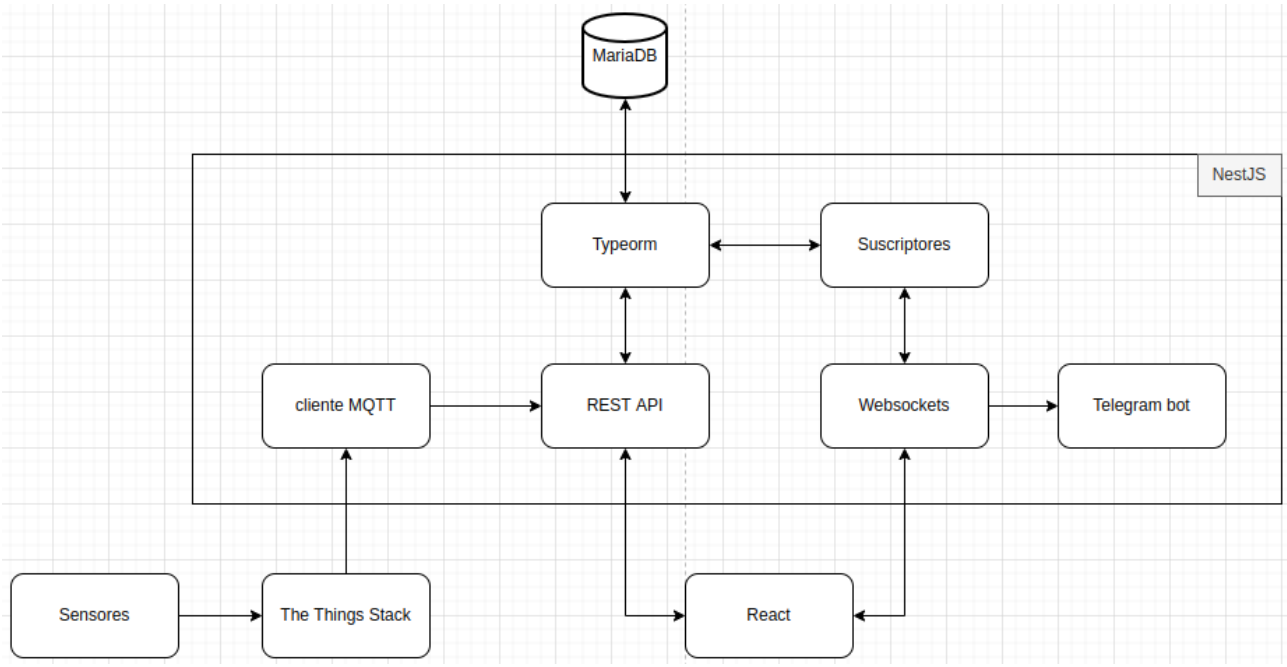


Figura 3.21: Diagrama de comunicación entre los diferentes componentes.

Cuando el sensor envía los datos, lo hace a través de LoRaWan con la configuración correcta para que un AP los pueda recibir. Este enviará los datos a TTS donde entra en los PF.

Cuando sale de estos formateadores, ya no es un *bytestring*, sino un formato legible por humanos, junto con otros datos de estadísticas que añade TTS. Después, estos son enviados sobre el MQTT *server* a nuestro cliente MQTT.

El cliente MQTT extrae la información de los sensores y el tiempo de envío del mismo y los envía al punto `/history/bulk` de nuestra API. Este *endpoint* es el único que está definido con esta opción y permite enviar un array de valores en vez de enviarlos uno a uno.

El REST API recibe este paquete y empieza a insertar los valores en la BBDD a través de TypeORM. Una vez que el dato se ha creado en la BBDD se lanza un suscriptor para informar a los WS del cambio y comprobar si lanzar una alarma.

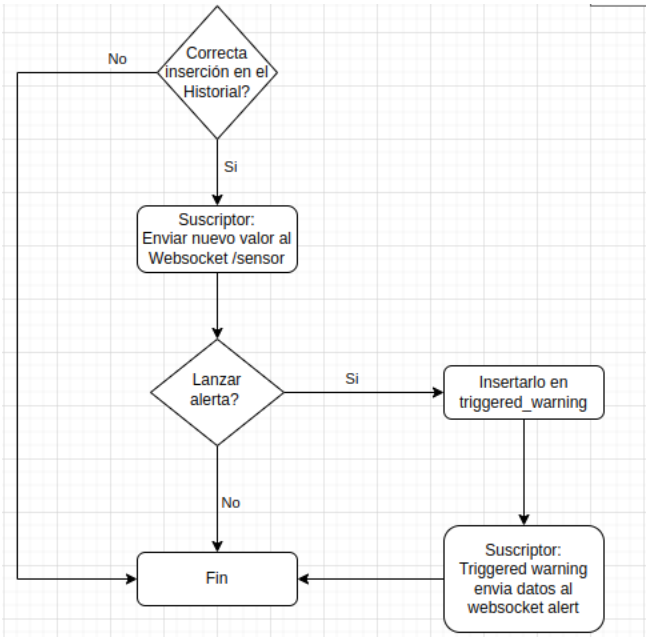


Figura 3.22: Lógica de ejecución de los suscriptores de la base de datos.

Dentro de los WS tenemos dos posibles puntos de acceso

- **/alert**: permite suscribirse para recibir todas las alertas.
- **/sensor**: no necesita ningún argumento para establecer la comunicación pero no empieza enviar datos automáticamente. Este requiere que el usuario envíe un mensaje como evento "subscribe", para definir qué datos quiere recibir. En el caso que el mensaje de subscribe no contenga ninguna información, se enviarán todos los datos recibidos a este cliente. Pero lo normal es, que el cliente especifique un dispositivo y un sensor dentro de este dispositivo, de los cuales quiere recibir datos. Otra posibilidad es que solo especifica el sensor. Esto permite recibir todos los datos de todos los dispositivos de un sensor concreto. Como podría ser, por ejemplo, la temperatura.

Por último, el bot de Telegram está suscrito al WS de alertas, cuando recibe una alerta, comprueba si existe uno o varios chats suscritos a esta alerta y, en caso afirmativo, envía un mensaje a todos ellos.

Capítulo 4

Conclusiones y líneas futuras

En este apartado nos gustaría presentar las conclusiones obtenidas durante el desarrollo del prototipo, y proponer algunas posibles líneas futuras para el proyecto.

4.1. Conclusiones

Durante el desarrollo de esta memoria hemos demostrado las diversas ventajas y funcionalidades que puede tener usar un sistema de estas características basado en la tecnología LoRaWan. Además, se podría predecir el impacto positivo en la calidad de vida de las personas, que puede llegar a tener la instalación de un sistema de control de constantes ambientales como el propuesto.

Adicionalmente, se puede observar que IoT no solo depende de los sensores, sino de todo el sistema de comunicación. Y, que este, ofrece una serie de ajustes que permiten adaptar el proyecto a las posibles necesidades futuras.

Por lo general, el proyecto se ha desarrollado haciendo especial hincapié en la modularidad y la adaptabilidad a futuro. Durante todo el desarrollo se ha estado intentando, en la medida de lo posible, predecir implementaciones futuras y refactorizando la aplicación para que estas queden cubiertas. Así, por ejemplo, se ha empezado el backend en Python, pero, para facilitar el mantenimiento y mejorar la calidad de los servicios prestados, se ha cambiado de tecnología.

Por último, se ha realizado un estudio exhaustivo de la tecnología LoRaWan y se han sentado las bases para la implementación de un proyecto práctico profesional, el cual podría llevarse a la realidad en las instalaciones de la Universidad de La Laguna.

4.2. Líneas futuras

En cuanto a las líneas futuras, hemos valorado algunos posibles casos de uso, en diferentes escalas de la aplicación. En primer lugar, como de momento solo está destinado a ser utilizado como prueba de conceptos, no se ha dado mucha importancia a implementar usuarios diferentes. Pero podría ser una buena idea implementar una personalización para cada usuario o grupos de usuarios y así tener más control sobre quienes reciben y pueden actuar sobre las alertas.

También se ha planteado la posibilidad de utilizar el proyecto para el control de recursos. Como puede ser, por ejemplo, un sistema de control de humedad en la tierra para alertar de cuando es necesario regar o incluso automatizar sistemas de riego. Escalando incluso más el concepto, podría llegar a estar relacionado con las denominadas Ciudades Inteligentes (CI). Este proyecto se ha realizado a pequeña escala teniendo en cuenta, únicamente, las instalaciones de la ULL. Pero estos mismos conceptos se podrían aplicar a una ciudad completa en la que se monitorizan parámetros ambientales para así poder mejorar la calidad de vida de sus habitantes. Es más, en muchas ciudades ya existe dicha monitorización. En esta misma línea, con referencia a las CI, se podría aplicar algoritmos de inteligencia artificial para poder predecir futuras alertas y problemas derivados de las variaciones de las constantes u optimizar los consumos de recursos de la ciudad.

Capítulo 5

Summary and Conclusions

In conclusion, during the development of this study and implementation, we have demonstrated the different advantages and functionalities that could be provided by a system of these characteristics, built over LoRaWan. Furthermore, we could assume that similar systems should have a positive impact in people's lives, since monitoring the environment should lead to a healthier population.

In addition, we have seen that IoT not only includes the installation of a sensor, but also, to properly work, it needs a complete communication system behind it. And, that current approximations, should be designed with future use cases in mind, so adaptability should be a key element.

So, naturally, this project has been developed with special emphasis on modularity and future adaptability. Throughout the development, we tried to keep everything as modular as possible and to predict future use cases. Refactoring or changing technologies if the prediction required such action. Thus, for example, the backend has started as a Python base project, but to facilitate maintenance and improve the quality of the services provided, the technology has been changed to NodeJS.

Finally, we carried out a study of the LoRaWan technology to lay the foundations for the implementation of a full scale and professional project of this kind, which could be brought to reality in the facilities of the University of La Laguna.

Capítulo 6

Presupuesto

Presupuesto sobre los costes de los recursos humanos y el trabajo realizado para este prototipo:

Tareas	Horas	Precio	Coste
Estudio de sensores adecuados	20	20 €/h	400€
Testeo inicial de los sensores, comunicación y de las configuraciones y payload formatters	40	20 €/h	800€
Estudio de tecnologías para el desarrollo	20	20 €/h	400€
Documentación	60	20 €/h	1200€
Desarrollo de la aplicación	210	20 €/h	4200€
Total	**350**		**7000,00€**

Tabla 6.1: Gastos de la mano de obra.

En resumen, para desarrollar el proyecto se han invertido 350 horas, con un precio final de 7000€

Para escalar el proyecto se puede calcular con un precio adicional por dispositivo (suponiendo que la instalación de los mismos ya se ha llevado a cabo)

Sensor	Precio dispositivo	Tiempo	Precio mano de obra	Total
AM307	275,89	30 min	20€/h	**285,89€**
VS121	241,80	1 hora	20€/h	**261,80€**
RAK7258	149,00	1 hora	20€/h	**169,00€**

Tabla 6.2: Precios de añadir otro dispositivo.

Bibliografía

[1] P. Wegner. “Global IoT market size grew 22 % in 2021 — these 16 factors affect the growth trajectory to 2027.” [Accessed 30-Jun-2022]. (2022), dirección: https://iot-analytics.com/iot-market-size/#:~:text=At%20this%20point%2C%20IoT%20Analytics,lowered%20from%20the%20previous%20year.

[2] Comisión Europea, *Ley de Datos: la Comisión propone medidas para una economía de los datos justa e innovadora*, [Accessed 04-Jul-2022], 2022. dirección: https://ec.europa.eu/commission/presscorner/detail/es/ip_22_1113.

[3] P. Wegner. “Global IoT spending to grow 24 % in 2021, led by investments in IoT software and IoT security.” [Accessed 30-Jun-2022]. (jun. de 2021), dirección: https://iot-analytics.com/2021-global-iot-spending-grow-24-percent/.

[4] behrtech, *6 Leading Types of IoT Wireless Tech and Their Best Use Cases*, [Accessed 03-Jun-2022], 2021. dirección: https://behrtech.com/blog/6-leading-types-of-iot-wireless-tech-and-their-best-use-cases/.

[5] S. Al-Sarawi, M. Anbar, K. Alieyan y M. Alzubaidi, “Internet of Things (IoT) communication protocols: Review,” en *2017 8th International Conference on Information Technology (ICIT)*, 2017, págs. 685-690. doi: 10.1109/ICITECH.2017.8079928.

[6] *LoRa Alliance*, [Accessed 04-Jul-2022]. dirección: https://lora-alliance.org/.

[7] J. Meijers, *TTN Mapper — ttnmapper.org*, [Accessed 28-Jun-2022]. dirección: https://ttnmapper.org/heatmap/.

[8] Milesight. “Milesight Ambience Monitoring Sensor.” [Accessed 28-Jun-2022]. (), dirección: https://www.milesight-iot.com/lorawan/sensor/am300/.

[9] ——, “Milesight AI Workplace Sensor.” [Accessed 28-Jun-2022]. (), dirección: https://www.milesight-iot.com/lorawan/sensor/vs121/.

[10] Dragino. “LHT65 LoRaWAN Temperature & Humidity Sensor.” [Accessed 28-Jun-2022]. (), dirección: https://www.dragino.com/products/temperature-humidity-sensor/item/151-lht65.html.

[11] T. T. Stack, *Cloud Integrations — thethingsindustries.com*, [Accessed 28-Jun-2022]. dirección: https://www.thethingsindustries.com/docs/integrations/cloud-integrations/.

[12] ——, *Adding Integrations — thethingsindustries.com*, [Accessed 28-Jun-2022]. dirección: https://www.thethingsindustries.com/docs/integrations/adding-integrations/.

[13] *MQTT*, [Accessed 04-Jul-2022]. dirección: https://mqtt.org/.

[14] Amazon, *¿Qué es una API?* [Accessed 03-Jul-2022]. dirección: https://aws.amazon.com/es/what-is/api/#:~:text=API%20significa%20%E2%80%9Cinterfaz%20de%20programaci%C3%B3n,de%20servicio%20entre%20dos%20aplicaciones..

[15] *Python*. dirección: https://www.python.org/.

[16] *paho-mqtt*. dirección: https://pypi.org/project/paho-mqtt/.

[17] *FastAPI*. dirección: https://fastapi.tiangolo.com/.

[18] *NodeJS*. dirección: https://nodejs.org/es/.

[19] *Typescript*. dirección: https://www.typescriptlang.org/.

[20] *NestJS*. dirección: https://nestjs.com/.

[21] IBM Cloud Education, *API REST*, [Accessed 04-Jul-2022], 2021. dirección: https://www.ibm.com/es-es/cloud/learn/rest-apis.

[22] *GraphQL*. dirección: https://graphql.org/.

[23] *MariaDB*. dirección: https://mariadb.org/.

[24] *PostgresSQL*. dirección: https://www.postgresql.org/.

[25] W. Crowell, *PostgreSQL vs. MariaDB: Features, Performance, Use Cases | OpenLogic by Perforce — openlogic.com*, [Accessed 28-Jun-2022], sep. de 2021. dirección: https://www.openlogic.com/blog/postgresql-vs-mariadb#:~:text=PostgreSQL%20outperforms%20MariaDB%20in%20regard,something%20not%20offered%20by%20PostgreSQL.

[26] *TypeORM*. dirección: https://typeorm.io/.

[27] *React*. dirección: https://es.reactjs.org/.

[28] *Vue*. dirección: https://vuejs.org/.

[29] *Angular*. dirección: https://angular.io/.

[30] J. Potter, *angular vs react vs vue | npm trends — npmtrends.com*, https://www.npmtrends.com/angular-vs-react-vs-vue, [Accessed 28-Jun-2022].

[31] J. de Carvalho Silva, J. J. P. C. Rodrigues, A. M. Alberti, P. Solic y A. L. L. Aquino, "LoRaWAN — A low power WAN protocol for Internet of Things: A review and opportunities," en *2017 2nd International Multidisciplinary Conference on Computer and Energy Science (SpliTech)*, 2017, págs. 1-6.

[32] B. Reynders y S. Pollin, "Chirp spread spectrum as a modulation technique for long range communication," en *2016 Symposium on Communications and Vehicular Technologies (SCVT)*, 2016, págs. 1-5. doi: 10.1109/SCVT.2016.7797659.

[33] J. Haxhibeqiri, E. De Poorter, I. Moerman y J. Hoebeke, "A survey of LoRaWAN for IoT: From technology to application," en, *Sensors (Basel)*, vol. 18, n.º 11, pág. 3995, nov. de 2018.

[34] M. Smart, *Wat is de technology efter LoRa-frekwinsje?* [Accessed 03-Jun-2022]. dirección: https://www.mokosmart.com/fy/lora-frequency/.

[35] S. Loukil, L. Fourati, A. Nayyar y K.-W.-A. Chee, "Analysis of LoRaWAN 1.0 and 1.1 Protocols Security Mechanisms," *Sensors*, vol. 22, pág. 3717, mayo de 2022. doi: 10.3390/s22103717.

[36] T. T. Network, *Regional Parameters — thethingsnetwork.org*, [Accessed 28-Jun-2022]. dirección: https://www.thethingsnetwork.org/docs/lorawan/regional-parameters/.

[37] M. Saelens, J. Hoebeke, A. Shahid y E. De Poorter, "Impact of EU duty cycle and transmission power limitations for sub-GHz LPWAN SRDs: an overview and future challenges," *EURASIP Journal on Wireless Communications and Networking*, vol. 2019, sep. de 2019. doi: 10.1186/s13638-019-1502-5.

[38] T. T. Network, *Duty Cycle — thethingsnetwork.org*, [Accessed 28-Jun-2022]. dirección: https://www.thethingsnetwork.org/docs/lorawan/duty-cycle/.

[39] ——, *Spreading Factors — thethingsnetwork.org*, [Accessed 28-Jun-2022]. dirección: https://www.thethingsnetwork.org/docs/lorawan/spreading-factors.

[40] E. Bassetti, *Enrico Bassetti — enricobassetti.it*, [Accessed 28-Jun-2022], feb. de 2018. dirección: https://www.enricobassetti.it/2018/02/lorawan-spreading-factor-allocation-in-a-multiple-gateway-environment/.

[41] A. Augustin, J. yi, T. H. Clausen y W. M. Townsley, "A Study of LoRa: Long Range & Low Power Networks for the Internet of Things," *Sensors*, vol. 16, n.º 9, pág. 1466, sep. de 2016. doi: 10.3390/s16091466. dirección: https://hal-polytechnique.archives-ouvertes.fr/hal-02263367.

[42] Qoitech, *How Spreading Factor affects LoRaWAN® device battery life — thethingsnetwork.org*, https://www.thethingsnetwork.org/article/how-spreading-factor-affects-lorawan-device-battery-life, [Accessed 28-Jun-2022].

[43] LoRa, *Understanding ADR | DEVELOPER PORTAL — lora-developers.semtech.com*, [Accessed 28-Jun-2022]. dirección: https://lora-developers.semtech.com/documentation/tech-papers-and-guides/understanding-adr.

[44] P. S. Cheong, J. Bergs, C. Hawinkel y J. Famaey, "Comparison of LoRaWAN classes and their power consumption," en *2017 IEEE Symposium on Communications and Vehicular Technology (SCVT)*, 2017, págs. 1-6. doi: 10.1109/SCVT.2017.8240313.

[45] T. T. Stack, *ABP vs OTAA — thethingsindustries.com*, [Accessed 28-Jun-2022]. dirección: https://www.thethingsindustries.com/docs/devices/abp-vs-otaa/.

[46] A. avbentem, *Airtime calculator for LoRaWANs*, [Accessed 03-Jul-2022]. dirección: https://avbentem.github.io/airtime-calculator/ttn/eu868/56.

[47] T. T. Stack, *Payload Formatters — thethingsindustries.com*, [Accessed 28-Jun-2022]. dirección: https://www.thethingsindustries.com/docs/integrations/payload-formatters/.

[48] Milesight. "AM307 & AM319 User Guide." [Accessed 28-Jun-2022]. (), dirección: https://resource.milesight-iot.com/milesight/document/am300-series-user-guide-en.pdf.

[49] *SwaggerUI*. dirección: https://swagger.io/tools/swagger-ui/.

[50] *phpMyAdmin*. dirección: https://www.phpmyadmin.net/.

[51] TypeORM, *Entity Listeners and Subscribers*, [Accessed 06-Jul-2022]. dirección: https://typeorm.biunav.com/en/listeners-and-subscribers.html.

[52] *Nestjsx Crud*. dirección: https://github.com/nestjsx/crud.

[53] M. Yali, *NestJSX Requests*, [Accessed 03-Jul-2022], 2019. dirección: https://github.com/nestjsx/crud/wiki/Requests#description.

[54] ——, *NestJSX Controllers*, [Accessed 03-Jul-2022], 2020. dirección: https://github.com/nestjsx/crud/wiki/Controllers#description.

[55] *socket.io*. dirección: https://socket.io/.

[56] *@nestjs/axios*. dirección: https://www.npmjs.com/package/@nestjs/axios.

[57] *node-telegram-bot-api*. dirección: https://www.npmjs.com/package/node-telegram-bot-api.

[58] *Material UI*. dirección: https://mui.com/material-ui/getting-started/overview/.

[59] *Recoil*. dirección: https://recoiljs.org/.

[60] *React Context*. dirección: https://es.reactjs.org/docs/context.html.

[61] *React Router*. dirección: https://reactrouter.com/.

[62] Javascript, *Bubbling and capturing*, [Accessed 04-Jul-2022], 2022. dirección: https://javascript.info/bubbling-and-capturing.

[63] *React ChartJS*. dirección: https://react-chartjs-2.js.org/.

[64] *recharts*. dirección: https://recharts.org/en-US/.

[65] *@emotion/css*. dirección: https://emotion.sh/docs/introduction.