

# Diseño, desarrollo y despliegue de una herramienta de simulación y ETL



Grado en Ingeniería Multimedia

## Trabajo Fin de Grado

Autor:  
Bruno García Escudero

Tutor/es:  
José Vicente Berná Martínez



Universitat d'Alacant  
Universidad de Alicante

## Resumen

Este trabajo se sitúa en el marco del proyecto *Smart University* de la Universidad de Alicante, cuyo objetivo es mejorar la calidad de vida en los campus universitarios mediante tecnologías avanzadas como el Internet de las Cosas (IoT) y el concepto de Smart Cities. En este contexto, se emplean sensores para recoger datos de la temperatura, calidad del aire y concentración de CO<sub>2</sub>, con el fin de optimizar la gestión de los recursos en los campus.

Dentro de este contexto, se ha creado una herramienta que simula los datos generados por estos sensores. La herramienta produce datos artificiales en tiempo real, como temperatura, humedad y niveles de CO<sub>2</sub>, que podrán ser enviados a Kunna, la plataforma centralizada para la visualización y análisis de datos. Esta herramienta se integrará en el ecosistema de Kunna bajo el nombre de *Kunna Genesis*, permitiendo la simulación de datos en situaciones hipotéticas y facilitando las pruebas y validaciones sin depender exclusivamente de datos reales.

## Resum

Aquest treball es situa en el marc del projecte *Smart University* de la Universitat d'Alacant, l'objectiu del qual és millorar la qualitat de vida als campus universitaris mitjançant tecnologies avançades com l'Internet de les Coses (IoT) i el concepte de Smart Cities. En aquest context, s'empren sensors per recollir dades de la temperatura, la qualitat de l'aire i la concentració de CO<sub>2</sub>, amb l'objectiu d'optimitzar la gestió dels recursos als campus.

Dins d'aquest context, s'ha creat una eina que simula les dades generades per aquests sensors. L'eina produeix dades artificials en temps real, com ara temperatura, humitat i nivells de CO<sub>2</sub>, que podran ser enviades a Kunna, la plataforma centralitzada per a la visualització i anàlisi de dades. Aquesta eina s'integrarà en l'ecosistema de Kunna sota el nom de *Kunna Genesis*, permetent la simulació de dades en situacions hipotètiques i facilitant les proves i validacions sense dependre exclusivament de dades reals.

## Summary

This work is part of the *Smart University* project at the University of Alicante, which aims to improve the quality of life on university campuses through advanced technologies such as the Internet of Things (IoT) and the concept of Smart Cities. In this context, sensors are used to collect data on temperature, air quality, and CO<sub>2</sub> concentration, with the goal of optimizing resource management on the campuses.

Within this framework, a tool has been created to simulate the data generated by these sensors. The tool produces artificial data in real-time, such as temperature, humidity, and CO<sub>2</sub> levels, which can be sent to Kunna, the centralized platform for data visualization and analysis. This tool will be integrated into the Kunna ecosystem under the name *Kunna Genesis*, enabling the simulation of data in hypothetical situations and facilitating testing and validation without relying solely on real data.

## Motivación, justificación y objetivo general

El proyecto surgió al finalizar el cuarto curso de la carrera, un año que supuso una gran experiencia de aprendizaje. Fue el primer curso en el que se aplicó la metodología ABP [21] (Aprendizaje Basado en Proyectos), lo que implicaba que todas las asignaturas estuvieran orientadas a un proyecto grupal. En mi caso, el itinerario fue sobre la gestión de contenidos, donde aplicamos todos los conocimientos adquiridos a lo largo de la carrera.

Esta forma de trabajo, junto con la oportunidad de conocer a grandes profesores comprometidos con su labor, me permitió conocer a José Vicente Berná. Él desempeñó un papel clave en el seguimiento y apoyo del proyecto grupal denominado *ARte*, en el que colaboré con mis compañeros, y que resultó ser todo un éxito.

Al finalizar el curso, y tras concentrar todas mis energías en ese proyecto, comencé a pensar en el tema del TFG. Fue entonces cuando José Vicente me propuso este proyecto, que me cautivó de inmediato. No solo me ofrecía la oportunidad perfecta para aplicar todo lo aprendido durante ese año, sino que además contribuiría al proyecto *Smart University*, en el que José Vicente y su equipo están trabajando.

Este proyecto me permitirá aplicar mis conocimientos en un entorno real, donde se requiere una solución concreta y donde tendremos que ir desarrollando el planteamiento inicial para encontrar una solución óptima. Espero que, en el futuro, sea útil para la universidad y que de alguna manera aporte valor. Sin duda, también será un comienzo en mi desarrollo profesional en este campo tan apasionante.

## Agradecimientos

En primer lugar, me gustaría agradecer a mi tutor, José Vicente Berná Martínez. Ha sido un gran aprendizaje y una experiencia muy enriquecedora trabajar contigo a lo largo de la carrera y durante el desarrollo del TFG.

Una carrera universitaria no solo implica desarrollo profesional, sino también crecimiento personal. Por ello, quiero agradecer la compañía y el apoyo de mis amigos de San Vicente. Esta etapa ha transcurrido junto a vosotros, y me la llevo para toda la vida: las experiencias y los aprendizajes compartidos han sido inolvidables.

Agradezco, asimismo, a todas aquellas personas con las que he coincidido durante la carrera. Gracias a vosotros, este proceso ha sido más llevadero y enriquecedor.

Por último, agradecer a mi familia por su apoyo incondicional desde el primer momento en que decidí salir de casa y embarcarme en esta aventura. Un agradecimiento especial a mis abuelos, por el interés y la ilusión con la que han vivido este proceso.

# Índice de contenidos

Resumen.....	1
Resum.....	2
Summary .....	3
Motivación, justificación y objetivo general .....	4
Agradecimientos .....	5
Índice de contenidos .....	6
Índice de figuras .....	8
Índice de tablas .....	11
1.    Introducción .....	12
2.    Planificación.....	13
3.    Estado del arte.....	15
3.1.    Tecnologías para el desarrollo .....	15
4.    Objetivos.....	17
5.    Metodología .....	18
6.    Implementación.....	21
Iteración inicial. Planificación y definición proyecto.....	21
6.1.    Iteración 1. Interfaces del proyecto.....	22
6.2.    Iteración 2. Preparación del entorno de implementación y creación de la base de datos local	28
6.3.    Iteración 3. Creación de ruta de autenticación y su componente en el frontend.....	33
6.4.    Iteración 4. Creación de ruta de gestión de usuarios y su componente en el frontend	35
6.5.    Iteración 5. Creación de ruta de localizaciones y su componente en el frontend .....	39
6.6.    Iteración 6. Creación de ruta de simulaciones y su componente en el frontend.....	42
6.7.    Iteración 7. Expansión de parámetros y nuevas funcionalidades de simulación .....	51
6.8.    Iteración 8. Integración y emisión de mensajes con MQTT.....	61
6.9.    Iteración 9. Creación de ruta de conexiones y su componente en el frontend .....	69

6.10. Iteración 10. Mejoras y refinamiento de la aplicación .....	76
1. Pruebas y validación.....	79
2. Resultados .....	80
3. Conclusiones y trabajo futuro .....	81
Referencias.....	82
Apéndice.....	84

# Índice de figuras

Figura 1. Diagrama ilustrando como el énfasis relativo en las distintas disciplinas cambia a lo largo del proyecto. (Fuente: <a href="https://es.wikipedia.oitorg/wiki/Proceso_unificado">https://es.wikipedia.oitorg/wiki/Proceso_unificado</a> ) .....	13
Figura 2. Panel de trabajo en Trello. (Fuente propia) .....	19
Figura 3. Rastreador de tareas en Clockify. (Fuente propia).....	19
Figura 4. Panel con el registro total de tareas en Clockify. (Fuente propia).....	20
Figura 5. Interfaz de inicio de sesión (Fuente propia).....	22
Figura 6. Interfaz de la página principal con la barra de navegación y opciones (Fuente propia)...	23
Figura 7. Interfaz de gestión de usuarios (Fuente propia) .....	24
Figura 8. Interfaz de la creación de nuevos usuarios (Fuente propia).....	25
Figura 9. Interfaz del apartado de localizaciones (Fuente propia).....	26
Figura 10. Interfaz de la creación de colecciones de localizaciones (Fuente propia) .....	27
Figura 11. Vista global de las interfaces (Fuente propia).....	27
Figura 12. Repositorio GitHb. Simulacion-ETL-TFG. (Fuente propia) .....	28
Figura 13. Pantalla de ejemplo de la plantilla Fuse (Fuente: <a href="https://fusetheme.com">https://fusetheme.com</a> ) .....	30
Figura 14. Estructura de carpetas del backend en Visual Studio Code (Fuente propia).....	31
Figura 15. Método authenticate() de auth.middleware.js (Fuente propia).....	32
Figura 16. Archivo connection.js (Fuente propia) .....	32
Figura 17. Prueba a la llamada login en Postman. (Fuente propia) .....	34
Figura 18. Página login implementada en la aplicación. (Fuente propia) .....	35
Figura 19. Página de gestión de usuarios de la aplicación. (Fuente propia) .....	37
Figura 20. Página de creación de usuario de la aplicación. (Fuente propia).....	38
Figura 21. Clase AdminGuard que verifica si el usuario es administrador. (Fuente propia).....	38
Figura 22. Rutas para usuarios autenticados. (Fuente propia) .....	39
Figura 23. Página de gestión de localizaciones de la aplicación. (Fuente propia) .....	41
Figura 24. Página de crear colección de localizaciones de la aplicación. (Fuente propia) .....	42
Figura 25. Ejemplo llamada POST a la colección Kunna Gen Test. (Fuente propia).....	43
Figura 26. Patrón JSON para generar simulaciones. (Fuente propia) .....	44
Figura 27. Ejemplo JSON final con datos generados. (Fuente propia).....	45
Figura 28. Página de gestión de simulaciones de la aplicación. (Fuente propia).....	47
Figura 29. Página de crear simulación de la aplicación. (Fuente propia).....	48
Figura 30. Método generateNewJson() encargado de generar el JSON final de la simulación. (Fuente propia) .....	50

Figura 31. Página de crear simulación de la aplicación con los nuevos campos. (Fuente propia) ..	53
Figura 32. Método simularInstantaneamente() encargado de generar simulaciones sin intervalo de tiempo. (Fuente propia) .....	55
Figura 33. Método simular() encargado de generar simulaciones en tiempo real. (Fuente propia)	57
Figura 34. Botones para generar simulaciones de manera instantánea o progresiva. (Fuente propia). ....	58
Figura 35. . Página de listado de simulaciones de la aplicación con los nuevos botones y funcionalidades. (Fuente propia) .....	59
Figura 36. Simulación generada instantáneamente a partir de una simulación de prueba. (Fuente propia) .....	60
Figura 37. Simulación en tiempo real a partir de una simulación de prueba. (Fuente propia) .....	60
Figura 38. Protocolo MQTT. ( <a href="https://www.paessler.com/es/it-explained/mqtt">https://www.paessler.com/es/it-explained/mqtt</a> ) .....	62
Figura 39. HiveMQ, servidor utilizado como entorno de pruebas de MQTT. ( <a href="https://www.foxon.cz/en/services/integrate-sw-platforms/hivemq-mqtt-broker">https://www.foxon.cz/en/services/integrate-sw-platforms/hivemq-mqtt-broker</a> ).....	63
Figura 40. Constructor encargado de establecer la conexión MQTT en mqtt.service. (Fuente propia) .....	64
Figura 41. Métodos sendMessage() y subscribeToTopic de mqtt.service. (Fuente propia) .....	65
Figura 42. Método sendMessageMqtt() encargado de enviar la simulación generada al servicio de MQTT. (Fuente propia).....	66
Figura 43. Llamada a la función sendMessageMqtt() al generar una nueva simlación. (Fuente propia) .....	66
Figura 44. Cluster creado en la página de HiveMQ para probar el envio de mensajes. (Fuente propia) .....	66
Figura 45. Web Socket Cliente de HiveMQ. (Fuente propia) .....	67
Figura 46. Web Socket Cliente de HiveMQ. (Fuente propia) .....	68
Figura 47. Página de gestión de conexiones de la aplicación. (Fuente propia) .....	71
Figura 48. Página de crear conexión de la aplicación. (Fuente propia) .....	72
Figura 49. Panel de simulacion con la nueva opción para seleccionar la conexión. (Fuente propia) .....	73
Figura 50. Método sendMessage() actualizado, pasando los datos de la conexión de la simulación al enviar el mensaje. (Fuente propia) .....	74
Figura 51. Método sendMessageMqtt que recibe los parámetros de la conexión, establece la conexión y envía el mensaje. (Fuente propia) .....	75
Figura 52. Prueba de envío de mensajes al mismo broker pero distinto tópico. (Fuente propia) ..	76
Figura 53. Página de simulaciones en ejecución. (Fuente propia).....	78



# Índice de tablas

Tabla 1. Planificación temporal TFG

15

# 1. Introducción

El proyecto Smart University [13], impulsado por la Universidad de Alicante, es una plataforma innovadora destinada a desarrollar un modelo de universidad que mejore la calidad de vida de la sociedad. Basado en los principios del Internet de las Cosas (IoT) y las Smart Cities , el objetivo es optimizar los recursos y la gestión del campus universitario mediante la monitorización de aspectos como la concentración de CO<sub>2</sub> en las aulas, la calidad del aire y el consumo eléctrico.

En este contexto, el proyecto se enfoca en mejorar la gestión y representación de datos provenientes de diferentes sensores. Estos datos, a menudo generados por sistemas externos y almacenados en archivos CSV, pueden ser complejos de manejar sin una herramienta adecuada. Por lo tanto, es crucial contar con simulaciones que generen datos artificiales para facilitar su visualización y análisis en diversos escenarios. Las simulaciones son fundamentales para prever comportamientos, realizar pruebas y validar sistemas sin depender únicamente de datos reales.

El objetivo principal de este proyecto es desarrollar una herramienta que permita generar simulaciones de datos y enviarlos automáticamente al software de Kunna [12]. Kunna actúa como una plataforma centralizada que gestiona datos de sensores, como temperatura, humedad y CO<sub>2</sub>, y facilita la creación de gráficos y mapas de calor para la visualización y análisis de estos datos. Esta herramienta mejorará la capacidad de análisis y la integración con Kunna, optimizando la representación de datos y contribuyendo a la eficiencia del proyecto Smart University.

## 2. Planificación

La planificación es esencial para el éxito del proyecto, ya que ofrece la estructura necesaria para cumplir los objetivos de manera eficiente. En la reunión inicial con el tutor José Vicente Berná, se definieron los objetivos y una planificación provisional para el proyecto, con el objetivo de entregarlo en la convocatoria C1, cuyo plazo final es diciembre de 2024.

Se consideró que el enfoque más adecuado sería adoptar un método iterativo y basado en sprints, ya que facilita una gestión ágil del proyecto, adaptándose a las necesidades y requisitos en el momento, mientras se avanza continuamente hacia los objetivos. Esta metodología Agile [24] se organiza en iteraciones, cada una de las cuales abarca fases de diseño, implementación, documentación y validación. Cada iteración tendrá una duración de una a dos semanas, dependiendo de la complejidad de las tareas a realizar y la disponibilidad.

El flujo de trabajo que se seguirá durante las iteraciones deberá parecerse al esquema mostrado en la siguiente figura.

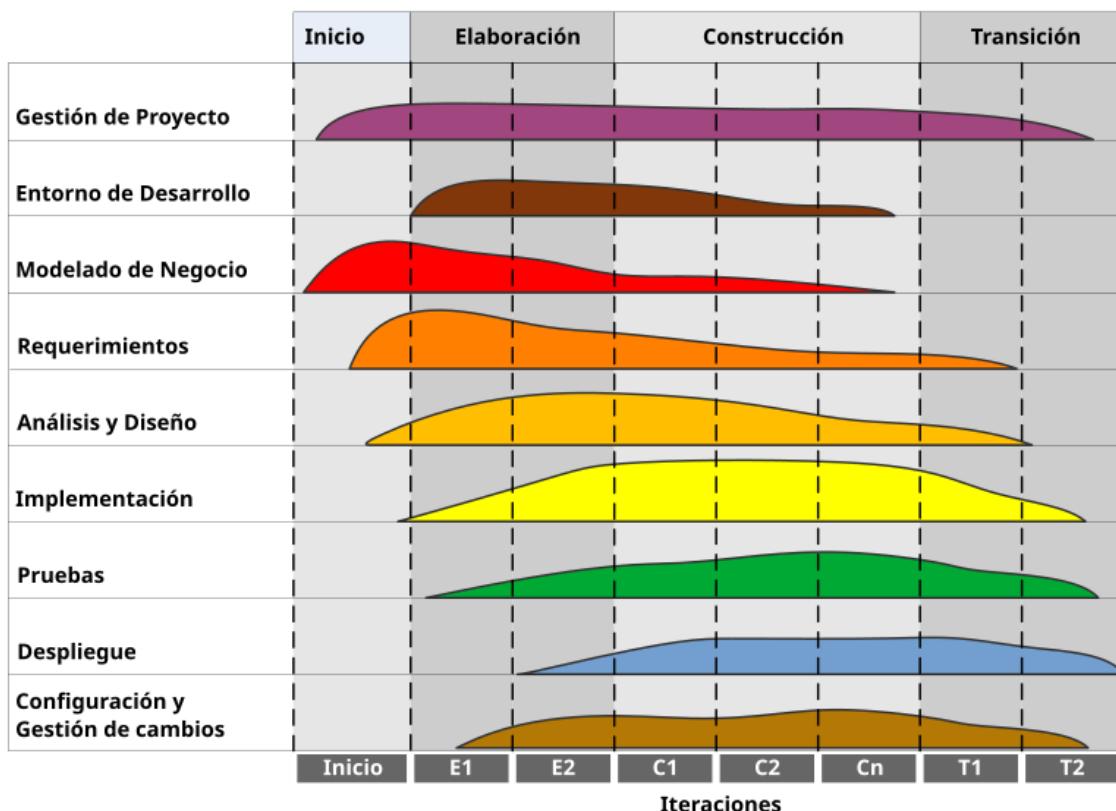


Figura 1. Diagrama ilustrando como el énfasis relativo en las distintas disciplinas cambia a lo largo del proyecto. (Fuente: [https://es.wikipedia.oiotrq/wiki/Proceso\\_unificado](https://es.wikipedia.oiotrq/wiki/Proceso_unificado))

Teniendo en cuenta lo anterior, la planificación temporal del proyecto se detalla en la siguiente tabla.

*Tabla 1. Planificación temporal TFG*

Iteraciones	Tiempo total	Fecha inicio
Iteración inicial. Planificación y definición proyecto	1 semana	14 agosto
Iteración 1. Interfaces del proyecto	2 semanas	21 agosto
Iteración 2. Preparación del entorno de implementación y creación de la base de datos local	1 semana	4 septiembre
Iteración 3. Creación de ruta de autenticación y su componente en el frontend	2 semanas	11 septiembre
Iteración 4. Creación de ruta de gestión de usuarios y su componente en el frontend	1 semana	24 septiembre
Iteración 5. Creación de ruta de localizaciones y su componente en el frontend	1 semana	1 octubre
Iteración 6. Creación de ruta de simulaciones y su componente en el frontend	1 semana	8 octubre
Iteración 7. Expansión de parámetros de simulación	2 semanas	15 octubre
Iteración 8. Integración y emisión de mensajes con MQTT	2 semanas	29 octubre
Iteración 9. Creación de ruta de conexiones y su componente en el frontend	1 semana	12 noviembre
Iteración 10. Mejoras y refinamiento de la aplicación	1 semana	19 noviembre
Iteración 11. Pruebas y validación	1 semana	26 noviembre
Iteración 12. Demo del proyecto y cierre de la memoria	1 semana	3 diciembre

### 3. Estado del arte.

Los sensores desempeñan un papel crucial en el contexto de Smart Cities y Smart University [13], ya que permiten la recopilación y transmisión de datos en tiempo real, lo que facilita la toma de decisiones para mejorar la eficiencia y sostenibilidad de los recursos. Sin embargo, el costo y el mantenimiento de los sensores reales pueden ser elevados. Aquí es donde entra la simulación de sensores, que permite replicar el comportamiento de estos dispositivos sin necesidad de hardware físico, lo que reduce costos y ofrece un entorno controlado para realizar pruebas antes de la implementación real.

En este contexto, el protocolo MQTT [14] (Message Queuing Telemetry Transport) juega un papel fundamental. MQTT es un protocolo de comunicación ligero y eficiente, diseñado especialmente para aplicaciones de Internet de las Cosas (IoT) [27]. Se utiliza para la transmisión de datos entre dispositivos, garantizando una comunicación rápida y eficiente, incluso en redes con ancho de banda limitado. Su diseño basado en la arquitectura de publicación y suscripción lo hace ideal para sistemas donde los sensores envían datos a servidores o aplicaciones de forma continua. Gracias a su bajo consumo de recursos y su capacidad para funcionar de manera confiable en redes inestables, MQTT es perfecto para escenarios de simulación de sensores.

Al utilizar MQTT en la simulación de sensores, se puede replicar de manera efectiva la interacción entre los dispositivos y el sistema central, permitiendo probar el flujo de datos y la integración sin tener que depender de hardware real. Esto es especialmente útil en entornos como Smart University, donde se puede simular el comportamiento de los sensores para gestionar el campus de manera más eficiente. A través de MQTT, los datos de los sensores simulados se pueden transmitir de forma instantánea y confiable, permitiendo realizar simulaciones realistas y optimizar el sistema de gestión del campus en tiempo real.

#### 3.1. Tecnologías para el desarrollo

Las herramientas tecnológicas utilizadas en este proyecto han sido seleccionadas para garantizar su integración con el ecosistema de Smart University [13], siendo además familiares para el estudiante gracias a su uso previo en el proyecto ABP [21] durante el cuarto curso del grado en Ingeniería Multimedia [25].

El backend del sistema se desarrolló utilizando Node.js en su versión 2.0.22, complementado con la biblioteca MySQL 2 3.6.2 para realizar consultas SQL, y Express 4.18.2 como marco para el

desarrollo de aplicaciones web. Para mejorar la seguridad y la gestión de configuraciones, se empleó Dotenv 16.3.1, permitiendo mantener datos sensibles, como credenciales y rutas, fuera de la lógica principal del código.

El frontend se implementó utilizando Angular 17.0.3, una de las versiones más recientes que garantiza un rendimiento optimizado y compatibilidad con las últimas herramientas. Para la interfaz de usuario, se combinaron Tailwind [26] CSS 3.3.5 y Bootstrap 5.3.0, logrando un diseño moderno, adaptable y altamente personalizable. Adicionalmente, se utilizaron Bootstrap Icons 1.10.5 para enriquecer la interfaz con iconografía estándar.

Por otra parte, el servidor local se configuró con XAMPP [6], alojando una base de datos relacional en MySQL para la gestión de datos de sensorización.

En cuanto al envío y recepción de mensajes en tiempo real, se utilizó el protocolo MQTT [14] en su versión 5.10.2, conocido por su ligereza y eficiencia en la transmisión de datos. Para las pruebas, se empleó HiveMQ [16] como broker, lo que facilitó la comunicación fluida y de baja latencia entre los dispositivos y el sistema. Esta solución se adapta perfectamente a los requerimientos del proyecto, ofreciendo una plataforma confiable y escalable para gestionar la interacción entre los sensores y el backend.

Por último, la gestión del proyecto y control de versiones se llevó a cabo en GitHub [2], utilizando un repositorio público alojado en la cuenta del estudiante para facilitar el acceso y la revisión por parte de los tutores y colaboradores.

## 4. Objetivos

El objetivo principal del proyecto es desarrollar una herramienta para generar simulaciones de datos artificiales provenientes de sensores y enviarlos a la plataforma Kunna. Esto facilitará la visualización y análisis de datos mediante gráficos y mapas de calor, además de permitir realizar pruebas y validar el funcionamiento sin depender de sensores reales.

A partir de este objetivo principal, surgen diversos objetivos secundarios, cada uno diseñado para satisfacer necesidades específicas y garantizar un funcionamiento óptimo de la herramienta.

En primer lugar, se contempla implementar un sistema de gestión de usuarios que garantice un acceso seguro y controlado a la herramienta. En lugar de permitir un registro abierto, el alta y la gestión de usuarios son realizadas exclusivamente por los administradores de la aplicación. Este enfoque asegura que solo personas autorizadas puedan acceder al sistema y operar la herramienta.

Otro aspecto importante es la usabilidad de la herramienta. Dado que manejar simulaciones involucra múltiples parámetros y configuraciones, resulta esencial contar con una interfaz clara e intuitiva para el usuario. La interfaz debe ofrecer alertas en caso de errores o configuraciones incorrectas, así como permitir una visualización constante del estado de las simulaciones. Además, incluirá un panel que permitirá pausar, reanudar y detener las simulaciones en tiempo real, brindando a los usuarios un control completo sobre el proceso.

Además, se plantea añadir un campo de conexiones que permita definir múltiples conexiones a distintos brokers y tópicos, proporcionando así el control total sobre el destino de las simulaciones. Esto ampliará la flexibilidad de la herramienta y facilitará su integración con Kunna y otros sistemas.

## 5. Metodología

La metodología elegida para el desarrollo del proyecto fue una metodología iterativa basada en sprints, tal como se explicó en la planificación. Esta metodología, permite una gestión flexible y ágil del proyecto, adaptándose a las necesidades y requisitos en cada momento, mientras se avanza de forma continua hacia los objetivos.

El trabajo se organizó en iteraciones, cada una abarcando fases de diseño, implementación, documentación y validación. Al finalizar cada iteración, se realiza una reunión con el tutor para analizar el progreso y definir los objetivos de la siguiente iteración. Estas reuniones son claves en el éxito del proyecto, ya que garantizan un avance constante y enfocado en la dirección correcta.

Para la correcta gestión y organización del proyecto, se va a hacer uso de 2 herramientas: Trello [18] y Clockify [19]. Ambas aplicaciones se usaron durante el cuarto curso de carrera, estando ya familiarizados con ellas.

Trello es una plataforma de gestión de proyectos que permite organizar y visualizar todas las tareas necesarias para el desarrollo del proyecto. En nuestro caso, se crearon tableros divididos en listas según el estado de las tareas: "Por hacer", "Iteración actual", "En revisión" y "Completado". Además, se usaron etiquetas para identificar a qué iteración pertenece cada tarea, asegurando un flujo de trabajo ordenado.

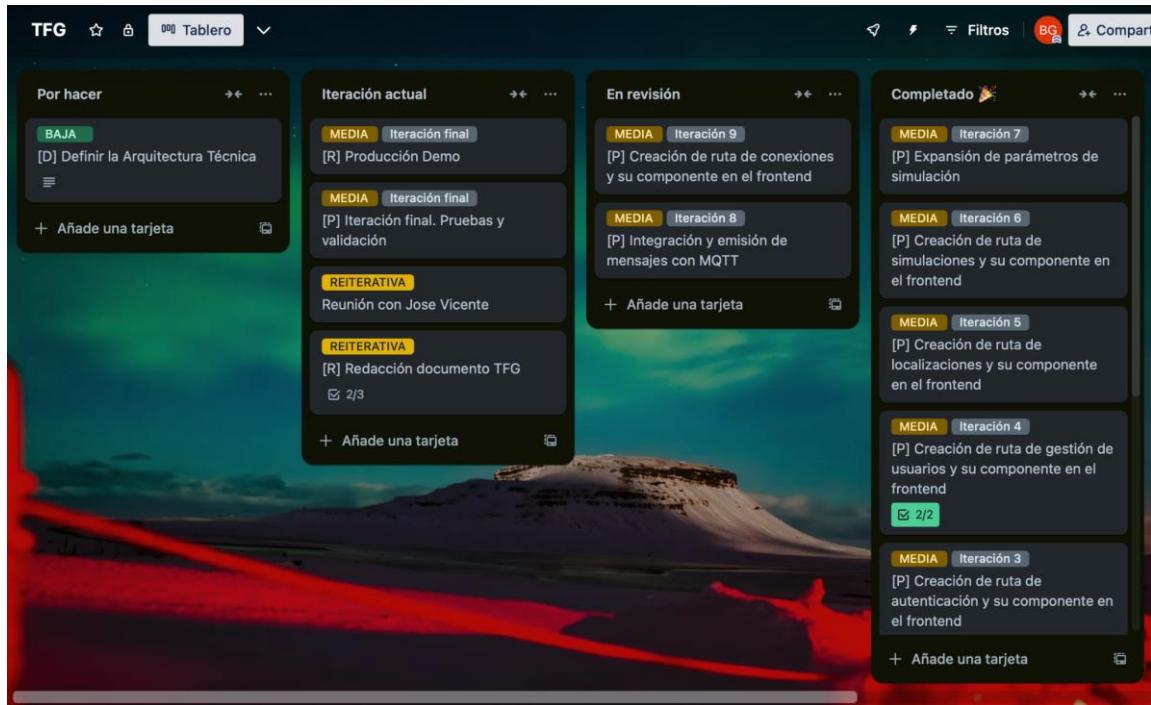


Figura 2. Panel de trabajo en Trello. (Fuente propia)

Clockify, por otro lado, es una herramienta que permite gestionar y registrar el tiempo dedicado a cada tarea de forma sencilla. Durante el desarrollo del proyecto, se utilizó para monitorizar el tiempo invertido en cada tarea, asociándose a su iteración correspondiente. Esto facilitó un seguimiento claro del progreso y ayudó a organizar el trabajo de manera más efectiva.

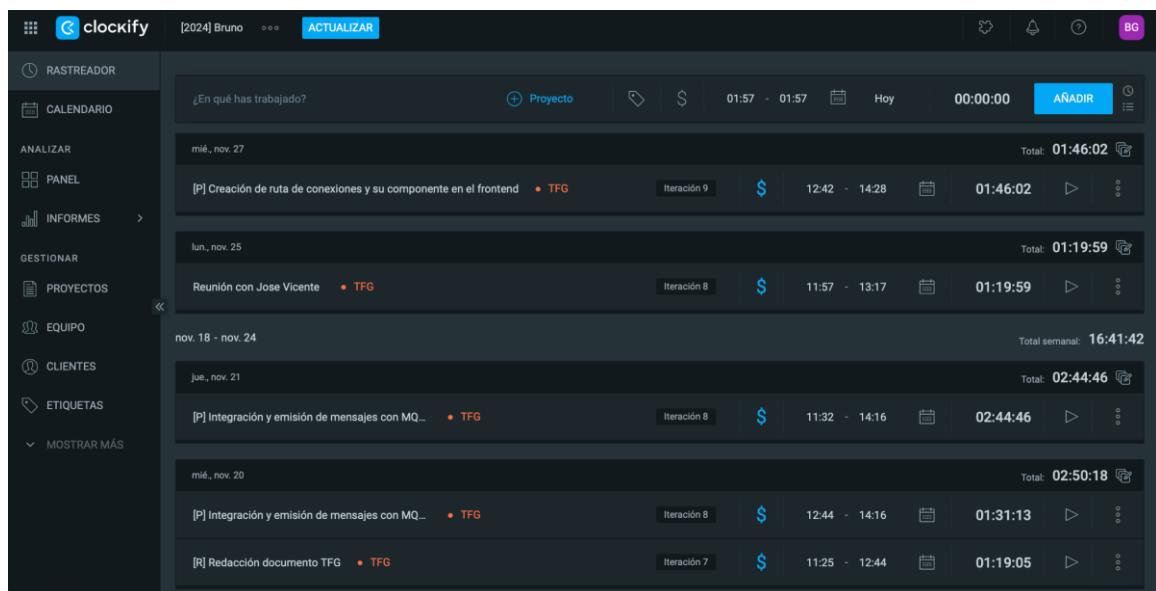


Figura 3. Rastreador de tareas en Clockify. (Fuente propia)

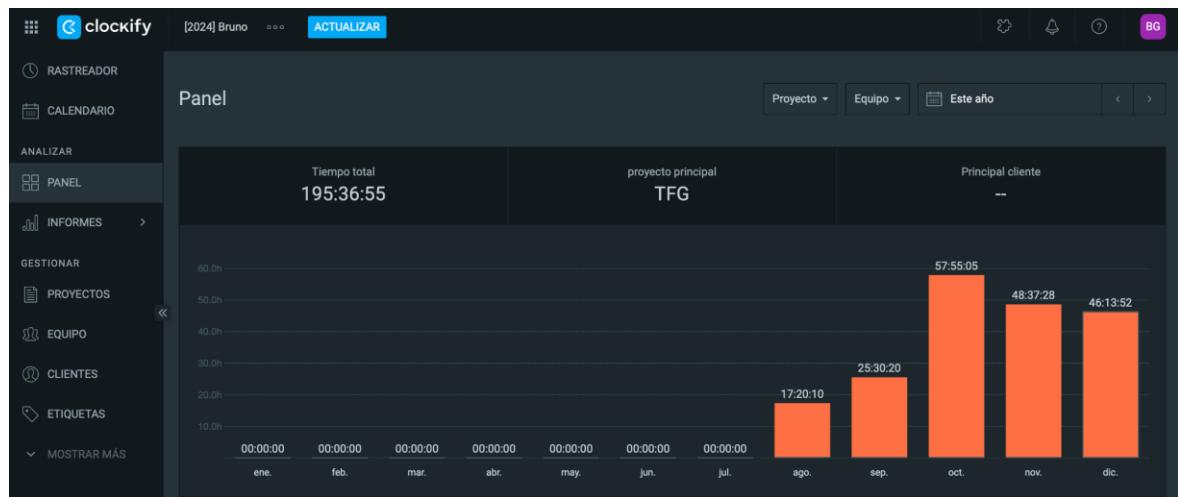


Figura 4. Panel con el registro total de tareas en Clockify. (Fuente propia)

## 6. Implementación

Como se mencionó anteriormente en el apartado de planificación, se consideró junto al tutor que la mejor opción era adoptar una metodología ágil basada en sprints.

Por ello, esta sección se divide en subapartados, en los que primero se detallarán los objetivos específicos de cada sprint, y luego se abordarán las distintas fases del proceso: análisis, diseño e implementación. Cabe destacar que, dependiendo de la naturaleza de cada iteración, no siempre se completarán todas las fases en cada una.

### Iteración inicial. Planificación y definición proyecto

Esta iteración se enfoca en la definición del proyecto y la planificación. En esta fase inicial, se llevó a cabo una reunión con el tutor José Vicente para validar la propuesta y establecer los objetivos y la estructura del proyecto.

El primer paso de esta iteración es identificar el problema principal: la necesidad de simulaciones que generen datos artificiales. Aunque la gestión y representación de datos generados por sensores, almacenados en archivos CSV y provenientes de sistemas externos, son tareas importantes, el desafío central radica en la falta de herramientas adecuadas para crear simulaciones efectivas. Estos datos incluyen información crítica como temperatura, humedad y CO<sub>2</sub>, y es fundamental poder trabajar con datos artificiales para visualizar y analizar estos datos en diversos escenarios.

Con la problemática definida, se concluye que es fundamental desarrollar una aplicación capaz de generar simulaciones de datos artificiales y enviarlos automáticamente a la API REST de Kunna. Kunna es una plataforma centralizada que gestiona datos de sensores y facilita la creación de gráficos y mapas de calor. La aplicación deberá optimizar la representación de estos datos, mejorar la integración con Kunna y facilitar la visualización en diferentes contextos. De esta manera, se potenciará la capacidad de análisis y se proporcionará una herramienta eficaz para el proyecto.

Además, en esta iteración se establece una planificación provisional mediante una metodología ágil basada en sprints, ya que esta metodología nos permitirá avanzar de manera iterativa, adaptando el proyecto a los requisitos que surjan a lo largo del proceso. La Iteración 0 sienta las bases del proyecto, asegurando una estructura organizada y una planificación efectiva para las siguientes fases de desarrollo.

## 6.1. Iteración 1. Interfaces del proyecto

Una vez definido el proyecto, en esta iteración pasamos al diseño de las primeras interfaces, para ayudarnos a concretar la idea y obtener una visión inicial de cómo funcionará el sistema. Entrando de esta manera en la fase de diseño.

### 6.1.1. Fase de diseño

Para realizar las interfaces, se utilizó la herramienta de Miro [1], que facilita la creación de interfaces de manera sencilla y detallada. En esta iteración, se comenzó por diseñar las interfaces iniciales, ya que el proyecto aún no estaba completamente definido. Este enfoque permitió establecer una base a partir de la cual se irán definiendo y ajustando las interfaces restantes según las necesidades que surjan durante el desarrollo. Se priorizaron las interfaces para el inicio de sesión, la gestión de usuarios y el panel de localizaciones, esenciales para el arranque del proyecto.

La primera interfaz desarrollada fue el inicio de sesión, que consiste en un formulario simple para acceder al sistema mediante un correo electrónico y una contraseña, como se muestra en la Figura 5. No se ha incluido una página de registro en el diseño, ya que la creación de nuevos usuarios es responsabilidad del usuario administrador.

The wireframe shows a login form titled 'Iniciar Sesión'. It contains two input fields: 'Correo electrónico' (Email) and 'Contraseña' (Password), each with a placeholder 'Agregar texto'. Below the password field is a checkbox labeled 'Recuérdame' (Remember me). At the bottom is a blue button labeled 'Iniciar sesión' (Login).

Figura 5. Interfaz de inicio de sesión (Fuente propia)

Una vez iniciado sesión, el usuario accede a la página principal, que corresponde al apartado de simulaciones (Figura 6). Dado que esta sección es la más compleja e importante, se decidió no abordarla en profundidad en esta iteración, sino dejar su desarrollo para etapas posteriores. En lugar de un diseño completo, se elaboró un boceto inicial que incluye el listado de simulaciones con sus datos, diversas acciones, así como un buscador y un botón para crear una nueva simulación.

Nombre	Total posiciones	Acciones
Simulación 1	13	
Simulación sensores	24	
Simulación rápida	17	

Figura 6. Interfaz de la página principal con la barra de navegación y opciones (Fuente propia)

A continuación, se trabajó en el apartado de gestión de usuarios. Esta sección facilita la administración de los usuarios del sistema, permitiendo crear nuevos, editar los existentes o eliminarlos. En la primera interfaz (Figura 7), se presenta un listado con el correo electrónico, el rol de cada usuario, y las opciones para editar o eliminar. Posteriormente, en una reunión con el tutor, se consideró la inclusión de un botón adicional para activar o desactivar temporalmente a un usuario específico, evitando así la necesidad de eliminarlo.

Correo electrónico	Rol	Acciones
tfe@gcloud.ua.es	Admin	
der6@gcloud.ua.es	Admin	
plo8@gcloud.ua.es	Usuario	

Figura 7. Interfaz de gestión de usuarios (Fuente propia)

Si se pulsa el botón "Nuevo", se accede a la página de creación de nuevos usuarios (Figura 8). Esta página presenta un formulario con campos para ingresar el correo electrónico, la contraseña y el rol del usuario. La contraseña se establece al momento de dar de alta a los usuarios, y se les permitirá cambiarla al iniciar sesión por primera vez. Además, se selecciona un rol que puede ser:

- Administrador: Tiene acceso a la gestión de usuarios, así como la capacidad de ver todas las simulaciones y localizaciones de todos los usuarios.
- Usuario: No tiene acceso a la gestión de usuarios y solo puede ver las simulaciones y localizaciones que ha creado personalmente.

Una vez que se ha completado el formulario y se pulsa el botón "Crear", si los datos introducidos son correctos, se procederá a la creación del nuevo usuario en el sistema. Posteriormente, en una reunión con el tutor, se propuso sustituir el campo de correo electrónico en el formulario por un nombre de usuario.

The screenshot shows a user interface for creating a new user. At the top, there are navigation links: 'Simulaciones', 'Localizaciones' (which is highlighted in blue), and 'Gestión usuarios'. On the right side, there are icons for language (Spanish flag) and user profile. The main area is titled 'Nuevo usuario'. It contains three input fields: 'Correo electrónico' (Email) with placeholder 'Agregar texto', 'Contraseña' (Password) with placeholder 'Agregar texto', and 'Rol' (Role) with a dropdown menu showing 'Admin'. To the right of the dropdown is a blue button with a '+' icon and the text 'Crear' (Create).

Figura 8. Interfaz de la creación de nuevos usuarios (Fuente propia)

Por último, se abordó el apartado de localizaciones. En esta sección, se podrán crear y gestionar listas de localizaciones que se importarán posteriormente a las simulaciones. Esto permite evitar la necesidad de ingresar los mismos datos una y otra vez para cada simulación, facilitando la reutilización de información previamente definida. La primera interfaz (Figura 18) muestra un listado de localizaciones con su información correspondiente, así como las opciones para editar o eliminar cada entrada. Además, se incluye un buscador para facilitar la búsqueda de localizaciones y un botón para crear nuevas.

Nombre	Total posiciones	Acciones
Edificios universidad	13	<input checked="" type="checkbox"/> <span style="color: red;">trash</span>
Recorrido carretera	24	<input checked="" type="checkbox"/> <span style="color: red;">trash</span>
Fuentes	17	<input checked="" type="checkbox"/> <span style="color: red;">trash</span>

Figura 9. Interfaz del apartado de localizaciones (Fuente propia)

Si se pulsa el botón "Nueva", se accede a la página de creación de nuevas localizaciones (Figura 10). Esta página presenta un formulario con los siguientes elementos:

- Nombre de la colección: Un campo para ingresar el nombre de la colección de localizaciones que se va a crear.
- Tabla de posiciones: Una tabla que permite introducir las coordenadas de cada localización. En cada fila de la tabla se deben ingresar:
  - o Latitud: La latitud de la localización.
  - o Longitud: La longitud de la localización.
  - o Alias: Un nombre o etiqueta para identificar la localización de manera más descriptiva.

Al final de la tabla, se encuentra un botón "+" que permite añadir una nueva fila para ingresar más localizaciones. Esto facilita la introducción de múltiples datos de manera organizada, permitiendo al usuario completar la tabla con la información necesaria para cada localización antes de guardar los cambios.

Una vez completado el formulario y pulsado el botón "Crear", si los datos introducidos son correctos, se procederá a la creación de la colección de localizaciones. Posteriormente, en una reunión con el tutor, se propuso añadir un campo de altura para cada localización, permitiendo así representar diferentes plantas dentro de un mismo edificio. Además, se consideró añadir un botón

al final de cada fila para eliminarla, y se evaluó la posibilidad de guardar el campo de posiciones en formato JSON [22] para facilitar su posterior tratamiento.

La captura de pantalla muestra la interfaz de usuario para la creación de una nueva colección de localizaciones. En la parte superior, hay un menú con las opciones "Simulaciones", "Localizaciones" (destacada en azul) y "Gestión usuarios". A la derecha del menú están los iconos de idioma (Español) y perfil de usuario. El título central es "Nueva colección". La sección "Nombre" tiene un cuadro de texto con el placeholder "Aregar texto". Abajo de esto, se titula "Posiciones" y muestra una tabla con tres columnas: "Latitud", "Longitud" y "Alias". La tabla tiene cinco filas vacías y un botón "+" para agregar más. En la parte inferior derecha, hay un botón azul con el icono de más y la palabra "Guardar".

Figura 10. Interfaz de la creación de colecciones de localizaciones (Fuente propia)

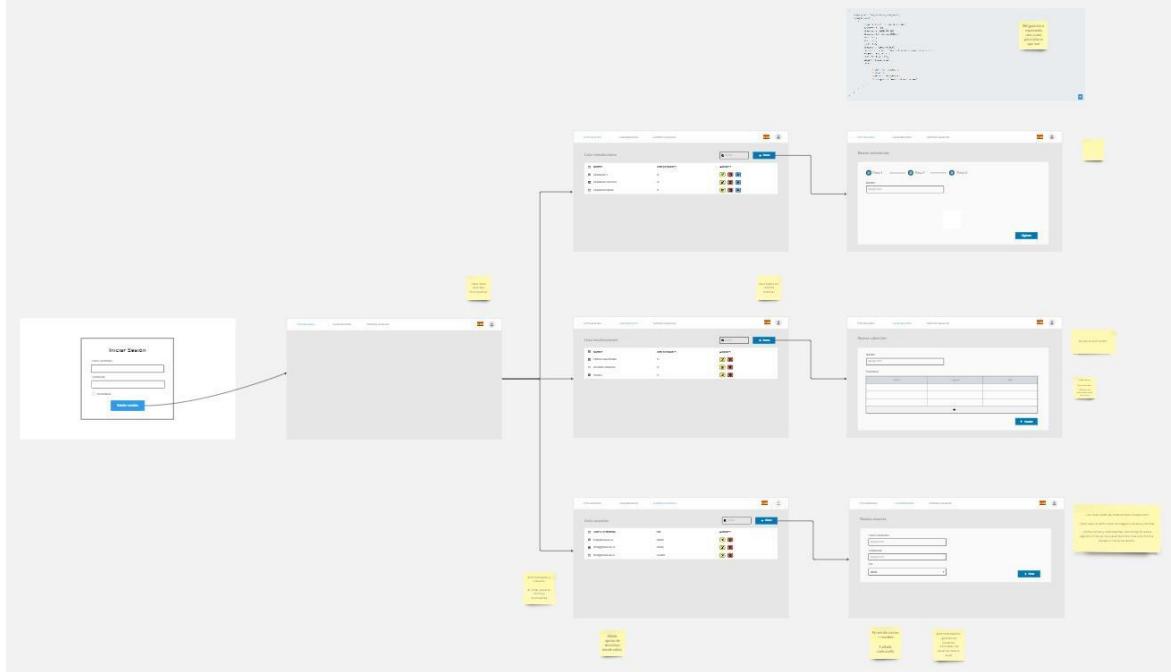


Figura 11. Vista global de las interfaces (Fuente propia)

## 6.2. Iteración 2. Preparación del entorno de implementación y creación de la base de datos local

Concluida la fase de diseño de interfaces y con las ideas más consolidadas, se inicia la fase de implementación. Esta iteración marca el comienzo del desarrollo del proyecto, y su objetivo es establecer las bases del proyecto. Se asegura que el entorno de desarrollo esté listo y que la base de datos esté configurada adecuadamente, lo que permitirá comenzar la programación de las funcionalidades y características previstas en el diseño.

### 6.2.1. Fase de implementación

La iteración comenzó con la creación de un repositorio en GitHub [2], que será la principal herramienta de gestión del proyecto. Para facilitar la revisión por parte de los tutores y colaboradores, el repositorio se ha configurado como público. Se puede acceder a él bajo el nombre Simulacion-ETL-TFG [3]. El repositorio cuenta con dos ramas principales: la rama master, que contiene el código final ya revisado, y la rama develop, donde se desarrollan y suben nuevas funcionalidades, pendientes de revisión.

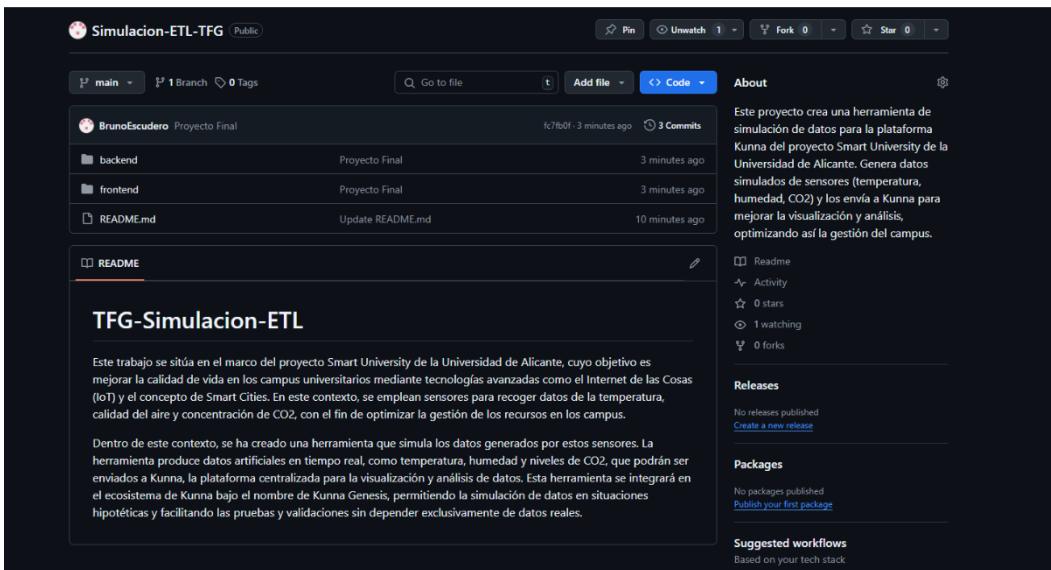


Figura 12. Repositorio GitHub. Simulacion-ETL-TFG. (Fuente propia)

Una vez creado el repositorio, se procedió a la creación del proyecto, que se divide en dos partes: backend y frontend. En el backend, se utiliza Node.js para gestionar la lógica del servidor, la conexión con la base de datos y las rutas API que permiten la comunicación entre el cliente y el servidor. Por otro lado, el frontend está desarrollado con Angular [20] e incluye todos los archivos y componentes necesarios para la interfaz de usuario y la interacción con el sistema.

Se eligió Visual Studio Code [4] como el editor de código por su interfaz fácil de usar y su buena conexión con GitHub, lo que facilita el trabajo y el seguimiento de cambios. También permite ejecutar comandos directamente en su terminal, lo que ayuda a iniciar el proyecto en Angular y Node.js. Para gestionar los repositorios de forma más sencilla, se utilizó GitHub Desktop [5], una herramienta que ofrece una interfaz gráfica para trabajar con los repositorios en GitHub.

En cuanto a la base de datos, se configuró un entorno local para la base de datos utilizando XAMPP [6], que combina un servidor Apache y MySQL/MariaDB, facilitando el desarrollo de aplicaciones web. Para gestionar las bases de datos, se emplearon PhpMyAdmin y HeidiSQL, que ofrecen interfaces gráficas para simplificar la conexión y administración. En esta fase, se creó una base de datos local en PhpMyAdmin con una tabla para almacenar información de usuarios, necesaria para el sistema de autenticación. Esta tabla incluye campos como ID, nombre de usuario, contraseña, rol (administrador o básico) y estado (activo o inactivo). Se dejarán las tablas adicionales para crear en futuras iteraciones según sea necesario.

A través de la terminal de Visual Studio Code, se llevó a cabo la instalación de todos los componentes necesarios para el backend del proyecto, que se desarrolla en Node.js [7], específicamente en la versión 20.10.0. Se utilizó el gestor de paquetes npm, el cual facilitó la instalación de bibliotecas clave. Entre las librerías instaladas se incluyen: bcrypt para la encriptación de contraseñas, dotenv para el manejo de variables de entorno, express para la creación del servidor web, jsonwebtoken para implementar la autenticación mediante tokens, y sequelize para facilitar la interacción con la base de datos.

Por otro lado, para el frontend, se instalaron las dependencias de Angular, como @angular/core, @angular/common, y @angular/material, todas en su versión 17.0.3. Además, se optó por utilizar la plantilla Fuse Angular [8], proporcionada por el tutor, que facilita el desarrollo gracias a su estructura bien organizada y componentes preconstruidos, permitiendo enfocarnos en la funcionalidad en lugar de en el CSS. Tras revisar su documentación, se procedió a la configuración inicial de la plantilla. La personalización de colores y layouts se dejó para iteraciones futuras.

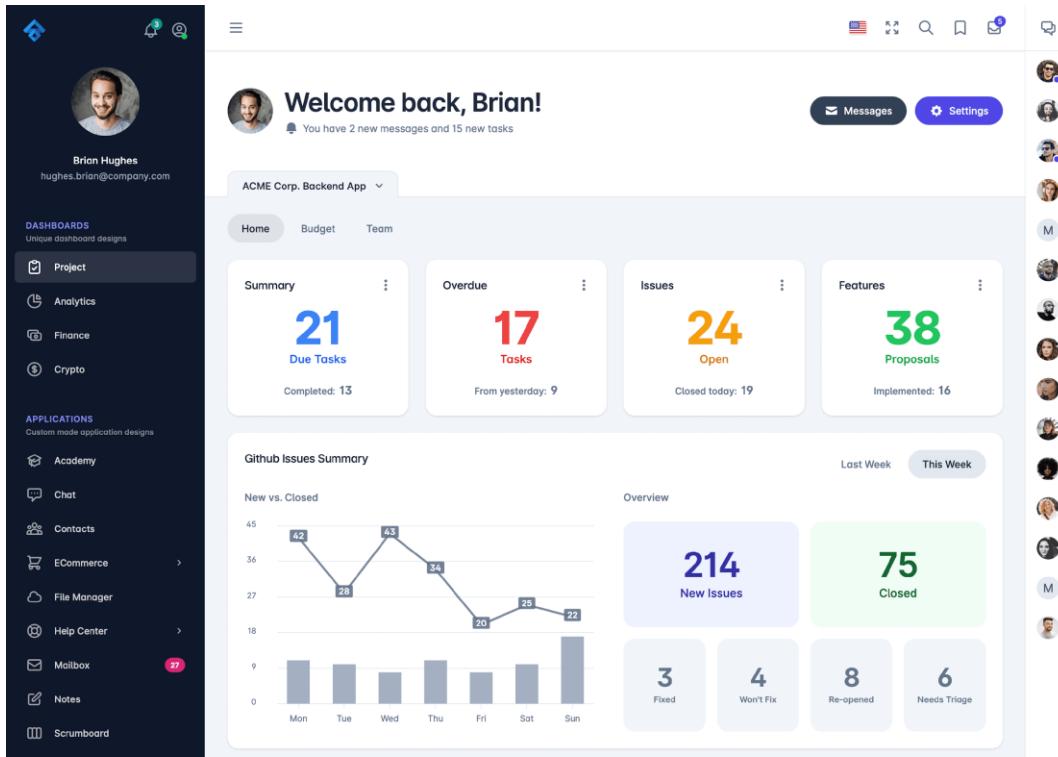


Figura 13. Pantalla de ejemplo de la plantilla Fuse (Fuente: <https://fusetheme.com>)

El desarrollo de la API comenzó con la creación del proyecto mediante el comando `npm init` y la instalación de ExpressJS [9] (v4.21.0) como framework para agilizar el proceso y evitar desarrollar todo desde cero. Luego, se configuró el archivo principal `index.js` para que la aplicación escuchara en el puerto 3000, y se añadió "nodemon" para facilitar el desarrollo al permitir que la API se reinicie automáticamente al realizar cambios en el código. El proyecto se organizó en tres directorios principales: routers, donde se definen las rutas; controllers, que contiene las funciones que gestionan las operaciones CRUD (post, get, put, delete); y models, que define el esquema de nuestros objetos.

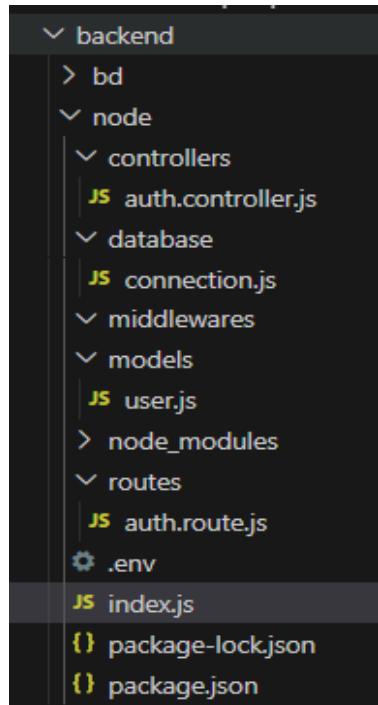


Figura 14. Estructura de carpetas del backend en Visual Studio Code (Fuente propia)

También se crearon carpetas adicionales para mantener el código más organizado. La carpeta database alberga el archivo connection.js (Figura 16), responsable de gestionar la conexión a la base de datos. Además, se creó la carpeta middlewares, que contiene funciones como la autenticación o el manejo de errores.

El método authenticate (Figura 15) es un middleware que protege rutas verificando tokens JWT. Extrae el token del encabezado Authorization de la solicitud, lo valida con una clave secreta (JWT\_SECRET) y, si es válido, asigna los datos del usuario decodificados a req.user para que puedan ser utilizados en las siguientes funciones. Si no se proporciona un token o este es inválido/expirado, responde con códigos de error 401 o 403 respectivamente, impidiendo el acceso no autorizado.

```

const jwt = require('jsonwebtoken');

const authenticate = (req, res, next) => {
    const token = req.headers.authorization?.split(' ')[1]; // Extraer el token del encabezado

    if (!token) {
        return res.status(401).json({ message: 'No se proporcionó un token.' });
    }

    try {
        // Verificar y decodificar el token
        const decoded = jwt.verify(token, process.env.JWT_SECRET);
        req.user = decoded; // Asigna los datos del usuario decodificados a req.user

        next(); // Continuar al siguiente middleware o controlador
    } catch (error) {
        return res.status(403).json({ message: 'Token inválido o expirado.' });
    }
};

module.exports = authenticate;

```

Figura 15. Método `authenticate()` de `auth.middleware.js` (Fuente propia).

Se utilizó Sequelize para gestionar la conexión con MariaDB de manera eficiente. El código importa Sequelize y dotenv para cargar las credenciales desde un archivo .env, protegiendo así la información sensible. Luego, se crea una instancia de Sequelize, a la cual se le pasan el nombre de la base de datos, usuario y contraseña mediante variables de entorno, y se especifican el host, dialecto (en este caso, MariaDB) y puerto de conexión.

```

const { Sequelize } = require('sequelize');
require('dotenv').config();

const sequelize = new Sequelize(
    process.env.DB_NAME,          // Nombre de la base de datos
    process.env.DB_USER,          // Usuario
    process.env.DB_PASSWORD,      // Contraseña
    {
        host: '127.0.0.1',
        dialect: 'mariadb',
        port: 3306,
    }
);

module.exports = sequelize;

```

Figura 16. Archivo `connection.js` (Fuente propia)

### 6.3. Iteración 3. Creación de ruta de autenticación y su componente en el frontend

En esta iteración, el objetivo principal fue la implementación de las primeras rutas API y su correspondiente integración en el frontend, comenzando con las rutas de login y registro en el backend. De esta manera, se dio continuidad a la fase de implementación iniciada en la iteración anterior.

#### 6.3.1. Fase de implementación

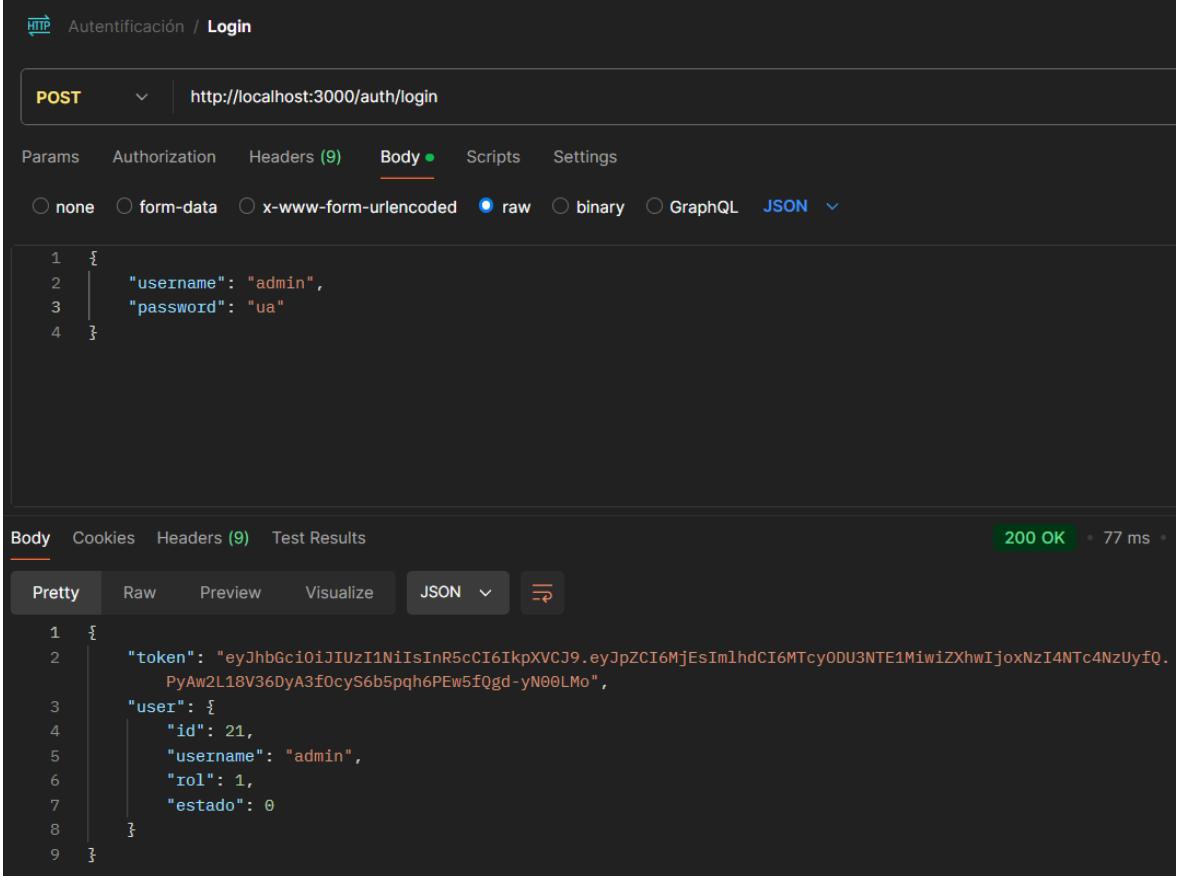
El primer paso fue definir las rutas de autenticación. Se creó una ruta base /auth que agrupa todas las operaciones relacionadas con la autenticación de usuarios. Dentro de esta ruta, se implementaron añadieron: /register para el registro de nuevos usuarios y /login para el inicio de sesión. Ambos aceptan solicitudes POST que envían los datos del usuario a través del cuerpo de la petición.

Para gestionar la lógica detrás de estas rutas, se desarrolló el archivo auth.controller.js, que contiene los métodos necesarios para el registro y la autenticación. El método register() es el encargado de gestionar la creación de nuevos usuarios. Recibe los datos del usuario (nombre de usuario, contraseña, rol y estado) como parámetros. Una vez recibidos, la contraseña es hasheada utilizando un algoritmo de encriptación para garantizar la seguridad. Posteriormente, se procede a crear el usuario en la base de datos.

Una vez finalizada la funcionalidad de registro, se procedió a implementar el método login(), que permite la autenticación de usuarios. Este método recibe el nombre de usuario y la contraseña proporcionados por el cliente y, en primer lugar, verifica si el usuario existe en la base de datos. Si el usuario es encontrado, la contraseña ingresada en texto plano se compara con la contraseña hasheada almacenada. Si la verificación es exitosa, se genera un token de autenticación, el cual es devuelto al cliente para manejar las sesiones de usuario.

Una vez creado todo lo necesario en el backend, procedemos a trabajar en el frontend y en el componente de inicio de sesión. Ayudándonos de la plantilla Fuse mencionada anteriormente, usamos el inicio de sesión predefinido que luego modificaremos a nuestro gusto. Creamos el componente desde la terminal con el comando ng generate component llamado sign-in, en el encontraremos el html y ts, a parte también creamos user.service.ts para conectar las peticiones api del back con el front.

Las pruebas se llevaron a cabo utilizando Postman [10], como se puede observar en las capturas de pantalla incluidas en la documentación.



The screenshot shows a Postman interface for an 'Autenticación / Login' collection. A POST request is made to `http://localhost:3000/auth/login`. The 'Body' tab is selected, showing a raw JSON payload:

```
1 {
2   "username": "admin",
3   "password": "ua"
4 }
```

The response status is `200 OK` with a response time of `77 ms`. The 'Pretty' tab shows the response body:

```
1 {
2   "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MjEsImlhCI6MTcyODU3NTc1MiwiZXhwIjoxNzI4NTc4NzUyfQ.PyAw2L18V36DyA3f0cyS6b5pqh6PEw5fQgd-yN00LMo",
3   "user": {
4     "id": 21,
5     "username": "admin",
6     "rol": 1,
7     "estado": 0
8   }
9 }
```

Figura 17. Prueba a la llamada login en Postman. (Fuente propia)

Con el backend ya implementado, se trabajó en el frontend, obteniendo como resultado la interfaz que se muestra en la Figura 25. Esta interfaz consiste en un formulario sencillo que incluye los campos de nombre de usuario y contraseña. Una vez que el usuario ha introducido sus datos, puede hacer clic en el botón de "Iniciar sesión", lo que genera una petición hacia el backend con los campos proporcionados.

Cuando el backend recibe la solicitud, valida la información enviada. Si no se encuentra ningún error, el inicio de sesión se realiza con éxito, redirigiendo automáticamente al usuario a la página principal de la aplicación, y guardando el token de autenticación para su uso posterior. Además, se ha implementado un AuthGuard en Angular para proteger las rutas del sistema, verificando si el usuario está autenticado mediante un token. Si no lo está, el guard redirige a la página de inicio de

sesión, asegurando que solo los usuarios autorizados accedan a áreas restringidas. Esta protección se aplica mediante la propiedad canActivate: [AuthGuard] en las rutas necesarias.

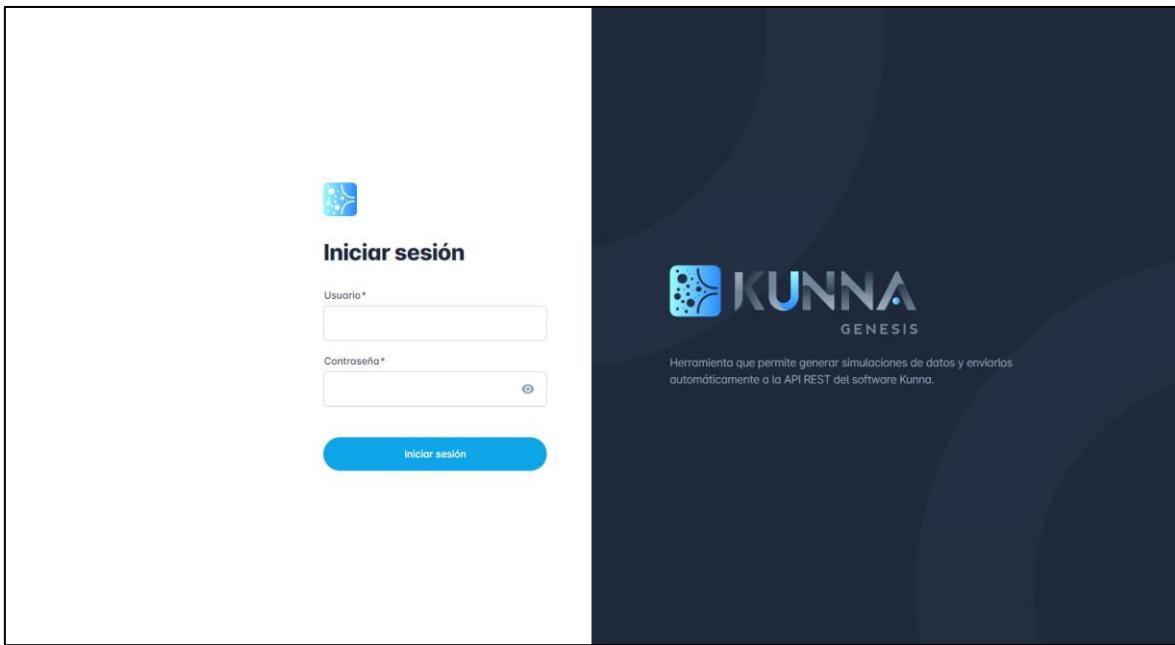


Figura 18. Página login implementada en la aplicación. (Fuente propia)

Es importante mencionar que en esta iteración no se implementó un componente en el frontend para el registro de usuarios, dejándolo para la iteración posterior. Esto se debe a que, en esta fase del proyecto, los usuarios no tienen la opción de registrarse por sí mismos. El registro de nuevos usuarios es gestionado exclusivamente por los administradores de la aplicación, quienes tienen la capacidad de darlos de alta a través del sistema de gestión de usuarios. Además, los métodos CRUD para editar, eliminar o listar usuarios también se implementarán en la siguiente iteración.

## 6.4. Iteración 4. Creación de ruta de gestión de usuarios y su componente en el frontend

En esta iteración, el objetivo principal fue la implementación de las rutas API y su correspondiente integración en el frontend, para realizar tanto el listado y gestión de los usuarios como la creación y edición de estos. Al igual que en iteraciones anteriores, se comenzó con una fase de análisis para obtener los requerimientos y posteriormente pasaremos a su implementación.

### 6.4.1. Fase de análisis

Dado que el sistema no cuenta con un apartado de registro de usuarios y la creación de cuentas es responsabilidad exclusiva de los administradores, se propone diseñar una página específica para la gestión de usuarios. Esta página permitirá a los administradores acceder a un listado completo de usuarios y realizar acciones como editarlos, eliminarlos, cambiar su estado (activo o inactivo) y crear nuevos usuarios. Estas acciones estarán restringidas únicamente a usuarios con rol de administrador.

Para proteger este acceso, nos decantamos por implementar un guard en Angular denominado `admin.guard.ts`, que verificará si el usuario tiene el rol de administrador. Si no lo tiene, será redirigido a la página principal, impidiéndole el acceso a la gestión de usuarios. Esta protección se aplicará mediante la propiedad `canActivate: [AdminGuard]`, asegurando que solo los administradores puedan acceder a estas funcionalidades.

En cuanto a las funcionalidades, ya se disponen de los métodos para registrar e iniciar sesión de usuarios, por lo que en esta iteración se contempla desarrollar métodos para editar, eliminar y obtener la lista de usuarios. La interfaz contará con una página inicial que mostrará el listado de usuarios junto con botones para editar, eliminar o crear nuevos usuarios. También se añadirá un buscador para facilitar la localización de usuarios en el listado. El botón de "Crear" dirigirá a otra página que contendrá un formulario para introducir el nombre de usuario, contraseña, rol (administrador o básico) y estado (activo o inactivo).

Por último, también se aprovechó para implementar un método de cierre de sesión, completando así el ciclo de autenticación y gestión de sesiones.

#### 8.5.2 Fase de implementación

Una vez definidos los requerimientos, se inició la implementación añadiendo nuevos métodos al controlador `auth.controller.js`, creado en la iteración anterior. Se implementó el método `deleteUser`, que recibe un ID como parámetro y elimina al usuario correspondiente si este existe. También se agregó el método `updateUser`, que toma el ID del usuario a editar y, en el cuerpo de la solicitud, los nuevos valores para su actualización. Además, se implementó el método `getAllUsers`, que devuelve la lista completa de usuarios, y `getUserById`, que permite obtener un usuario específico mediante su ID. Finalmente, se añadieron estas rutas a `auth.route.js` y se probó su correcto funcionamiento en Postman.

Con las rutas definidas y en funcionamiento, es necesario implementar un servicio que devuelva todos los usuarios registrados en la base de datos. Para ello, se aprovechó el servicio de autenticación (`auth.service.ts`) creado en la iteración anterior, al que se añadieron las funciones

CRUD que faltaban, como getAllUsers, deleteUser y updateUser. Además, se implementó un método signOut que elimina el token del almacenamiento local, establece la variable \_authenticated en false y redirige al usuario a la página principal.

Una vez realizadas estas implementaciones, se procede a la creación del componente y sus interfaces, denominado Usuarios. La página inicial (Figura 19) contendrá una tabla con cinco filas: las cuatro primeras mostrarán la información de cada usuario (ID, nombre de usuario, rol y estado), y la última incluirá dos botones: uno para editar, que redirige al componente correspondiente, y otro para eliminar, que al ser pulsado lanzará una llamada a deleteUser del servicio. También se incluirá un buscador para localizar un usuario específico en el listado, así como un botón de "Crear", que llevará al usuario a su correspondiente componente.

The screenshot shows a web application interface for managing users. At the top, there is a navigation bar with icons for Simulaciones, Localizaciones, and Gestión usuarios, and a user account icon labeled 'admin'. Below the navigation is a breadcrumb trail: 'Gestión usuarios > Inicio > Gestión de usuarios'. A search bar with placeholder 'Buscar...' and a blue 'Nuevo' button are also present. The main content area is titled 'Lista usuarios' and displays a table with four rows of user data:

ID	USUARIO	ROL	ESTADO	ACCIONES
24	admin	Administrador	Activo	
25	user1	Básico	Inactivo	
26	user 2	Básico	Activo	
27	user 3	Básico	Activo	

At the bottom left of the page, there is a small footer text: 'Genesis © 2024'.

Figura 19. Página de gestión de usuarios de la aplicación. (Fuente propia)

Los componentes de edición y creación compartirán el mismo formulario. En el caso de la creación, el formulario estará vacío y llamará a la función signUp del servicio, mientras que, en la edición, el formulario se completará con los datos correspondientes del usuario y llamará a la función updateUser, pasándole el ID del usuario.

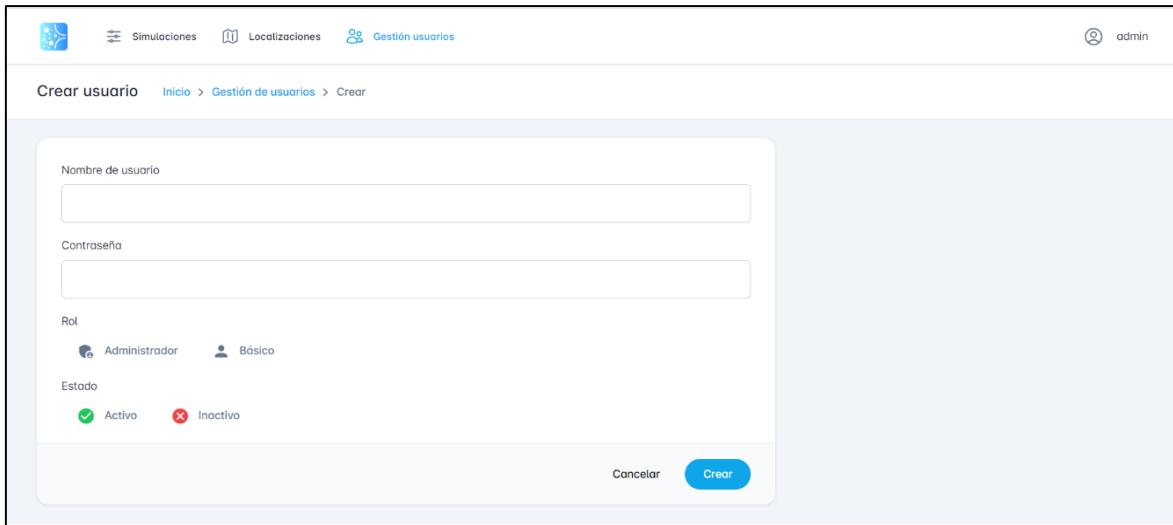


Figura 20. Página de creación de usuario de la aplicación. (Fuente propia)

Una vez creados los componentes y habiendo probado su correcto funcionamiento, el siguiente paso es permitir el acceso a esta página solo a administradores. Para ello, como se mencionó en la fase de análisis, se implementó un guard en Angular denominado admin.guard.ts (Figura 21), que verifica si el usuario tiene el rol de administrador llamando a la función isAdmin() del authService. Si no cumple este requisito, el sistema lo redirige a la página principal, evitando que acceda a la gestión.

```

@Injectable({
  providedIn: 'root'
})
export class AdminGuard implements CanActivate {
  constructor(private authService: AuthService, private router: Router) {}

  canActivate(): boolean {
    // Verificar si el usuario tiene rol de administrador
    if (this.authService.isAdmin()) {
      return true;
    }

    // Si no es admin, redirigir a la página principal
    this.router.navigate(['']);
    return false;
  }
}

```

Figura 21. Clase AdminGuard que verifica si el usuario es administrador. (Fuente propia)

Finalmente, esta protección se aplicó en la ruta "gestión\_usuarios" mediante la propiedad canActivate: [AdminGuard], garantizando que solo los administradores puedan acceder a estas funcionalidades. Además, se implementó un AuthGuard para restringir el acceso a las páginas

destinadas a usuarios autenticados, aplicando la propiedad canActivate: [AuthGuard] en las rutas correspondientes.

```
// Auth routes
{
  path: '',
  canActivate: [AuthGuard], // AuthGuard
  canActivateChild: [AuthGuard],
  component: LayoutComponent,
  data: {
    layout: 'modern'
  },
  resolve: {
    initialData: initialDataResolver
  },
  children: [
    {path: 'gestion_usuarios', loadChildren: () => import('app/modules/pages/usuarios/home.routes'), canActivate: [AdminGuard]},
    {path: 'example', loadChildren: () => import('app/modules/pages/example/example.routes')},
    {path: 'localizaciones', loadChildren: () => import('app/modules/pages/localizaciones/localizaciones.routes')},
    {path: 'simulaciones', loadChildren: () => import('app/modules/pages/simulaciones/simulaciones.routes')}
  ]
}
```

Figura 22. Rutas para usuarios autentificados. (Fuente propia)

## 6.5. Iteración 5. Creación de ruta de localizaciones y su componente en el frontend

En esta iteración, el objetivo principal fue la implementación de las rutas API y su correspondiente integración en el frontend, para realizar todos los métodos CRUD correspondientes a las colecciones de localizaciones. Al igual que en iteraciones anteriores, se comenzó con una fase de análisis para obtener los requerimientos y posteriormente pasaremos a su implementación.

### 6.5.1. Fase de análisis

Esta funcionalidad surge de la necesidad de facilitar el proceso de simulación al permitir la importación de colecciones de localizaciones. De esta manera, evitamos tener que introducir manualmente todas las ubicaciones cada vez que deseamos realizar una simulación, y en su lugar, podremos seleccionar y cargar directamente una lista de colecciones desde el menú. Cada colección estará compuesta por un nombre y un array de coordenadas que incluyen información sobre la longitud, latitud, cota y alias. Esto permitirá, por ejemplo, crear una colección de los principales edificios del campus universitario o de las localizaciones donde se encuentran los sensores de temperatura, optimizando así el manejo de datos en las simulaciones.

Por lo tanto, se plantea la necesidad de implementar las llamadas CRUD para gestionar las colecciones de localizaciones, lo que incluye crear, editar, eliminar y obtener un listado de las mismas. Cada colección estará compuesta por un nombre y un array de localizaciones que

almacenará los parámetros de longitud, latitud, cota y alias en formato JSON. Además, se considerará la creación de una clave foránea que refiera al usuario, permitiendo asociar cada localización con el usuario que la creó.

Es relevante destacar que los usuarios básicos tendrán acceso restringido, pudiendo ver y manipular únicamente las localizaciones que ellos mismos hayan creado, mientras que los administradores podrán acceder a todas las colecciones.

En cuanto a la interfaz, se propone que sea similar a las existentes, con una página inicial que muestre el listado de localizaciones, acompañada de botones para editar, eliminar y crear nuevas colecciones. También se incluirá un buscador para facilitar la búsqueda. La página de creación contará con un formulario para introducir el nombre de la colección y una tabla que incluya cuatro columnas: longitud, latitud, cota y alias. Además, de botones para añadir filas adicionales y eliminar filas existentes.

#### 6.5.2. Fase de implementación

Una vez definidos los requerimientos, se inició la implementación, comenzando por el backend con la creación de un modelo y un controlador para las localizaciones. El modelo incluirá un campo de tipo string para el nombre de la colección, un objeto JSON que almacenará las coordenadas como un array, y una clave foránea denominada userId, que hará referencia al usuario que ha creado la colección. Esta relación se establece utilizando Locations.belongsTo(User, { foreignKey: 'userId' }).

En cuanto al controlador, se desarrollarán los siguientes métodos:

- newLocation(): Este método se encargará de crear una nueva localización. Recibirá en el cuerpo de la solicitud el nombre de la colección y el JSON con las coordenadas, además de recuperar el ID del usuario desde el token de autenticación.
- deleteLocation(): Este método recibirá un ID como parámetro y, si la localización existe, procederá a eliminarla de la lista.
- getAllLocations(): Este método obtendrá el rol y el ID del usuario. Si el usuario es un administrador, se devolverán todas las localizaciones. En caso contrario, si el usuario tiene un rol básico, se filtrarán las localizaciones por su id, devolviendo únicamente aquellas que ha creado.
- getLocationById(): Este método recibirá un ID como parámetro y devolverá la localización correspondiente a ese ID.

Una vez realizadas estas implementaciones, se procede a la creación del componente y sus interfaces, denominado Localizaciones. La página inicial (Figura 23) contendrá una tabla con cuatro

filas: las tres primeras mostrarán la información de cada localización (ID, nombre, número de coordenadas), y la última incluirá dos botones: uno para editar, que redirige al componente correspondiente, y otro para eliminar, que al ser pulsado lanzará una llamada a deleteLocation del servicio. También se incluirá un buscador para localizar una localización específica en el listado, así como un botón de "Crear", que llevará al usuario a la página de crear localización.

The screenshot shows a web application interface for managing locations. At the top, there is a navigation bar with icons for Simulaciones, Localizaciones (selected), and Gestión usuarios, along with a user icon labeled 'admin'. Below the navigation is a breadcrumb trail: Localizaciones > Localizaciones. A search bar and a 'Nuevo' (New) button are also present. The main content area is titled 'Lista localizaciones' and contains a table with four rows of data:

ID	NOMBRE	NÚMERO COORDENADAS	ACCIONES
16	Edificios EPS	1	
17	Sensores CO2	3	
18	Sensores Humedad	2	
19	Aulas informática	2	

At the bottom left of the content area, there is a small text 'Genesis © 2024'.

Figura 23. Página de gestión de localizaciones de la aplicación. (Fuente propia)

Los componentes de edición y creación compartirán el mismo formulario. En el caso de la creación, el formulario estará vacío y llamará a la función newLocation del servicio, mientras que, en la edición, el formulario se completará con los datos correspondientes y llamará a la función updateLocation, pasándole el ID de la colección. Como se puede observar en la Figura 31, el formulario incluye una tabla para introducir las coordenadas. En esta tabla, el usuario podrá agregar nuevas filas con el botón "Aregar coordenadas" y eliminar filas específicas mediante un ícono rojo de eliminación, que eliminará la fila seleccionada al pulsarlo.

The screenshot shows a user interface for creating a location collection. At the top, there are navigation links: 'Simulaciones', 'Localizaciones', and 'Gestión usuarios'. On the right, there is a user icon labeled 'admin'. Below the navigation, the title 'Crear colección' is followed by a breadcrumb trail: 'Inicio > Localizaciones > Crear'. The main form has a 'Nombre' field (empty) and a 'Coordenadas' section. This section contains a table with four columns: 'Latitud', 'Longitud', 'Altura', and 'Alias'. There are two rows in the table, each with a small red trash can icon in the 'Alias' column. Below the table is a button '+ Agregar Coordenada' (Add Coordinate). At the bottom of the form are 'Cancelar' and 'Crear' buttons.

Figura 24. Página de crear colección de localizaciones de la aplicación. (Fuente propia)

Una vez creados los componentes y comprobado su funcionamiento, el siguiente paso fue gestionar el acceso según el rol de cada usuario. Como mencionamos anteriormente, los usuarios básicos solo podrán ver y manipular las localizaciones que ellos mismos hayan creado, mientras que los administradores tendrán acceso a todas las colecciones. No fue necesario realizar cambios adicionales en este aspecto, ya que el método `getAllLocations` estaba diseñado para filtrar las localizaciones en función del rol: si el usuario es administrador, se devuelven todas; si es básico, se filtran por su ID, mostrando solo las suyas.

Además, al igual que en otras rutas del sistema, se implementó un AuthGuard para restringir el acceso a usuarios no autenticados, aplicando la propiedad `canActivate: [AuthGuard]` en las rutas correspondientes.

## 6.6. Iteración 6. Creación de ruta de simulaciones y su componente en el frontend

En esta iteración, el objetivo principal fue implementar las rutas API y su integración con el frontend para gestionar todos los métodos CRUD relacionados con las simulaciones. Esta funcionalidad es clave para la aplicación, permitiendo generar simulaciones y prepararlas para su posterior envío al software de Kunna, donde serán visualizadas. El foco de esta fase está en la creación y visualización de simulaciones, generando los datos simulados y mostrándolos en pantalla. El envío de estos datos

a través del protocolo MQTT mediante Mosquitto[11] se abordará en siguientes iteraciones. Al igual que en fases anteriores, comenzamos con un análisis de los requerimientos antes de pasar a la implementación.

#### 6.6.1. Fase de análisis

Esta funcionalidad es el núcleo de la aplicación y su implementación ha pasado por varias fases y enfoques. Inicialmente, no estaba claro cómo se llevaría a cabo, por lo que no se definió una interfaz desde el principio, permitiendo ajustar los requisitos a medida que avanzaba el desarrollo.

En un primer planteamiento, se consideró que los datos simulados se enviarían a través de la API de Smart University mediante una petición POST, que incluiría la autorización necesaria para añadir un elemento a una colección específica. En la Figura 32 se muestra un ejemplo de esta llamada POST, donde el campo topic\_name corresponde al nombre de la colección a la que queremos enviar los datos, seguido de un objeto que contiene todos los datos simulados. Para su prueba, se dispone de una colección llamada “Kunna Gen Test”, a la que se pueden enviar los datos simulados para su visualización en el software de Kunna.

```
{
  "topic_name": "organization_name_topic",
  "arrayobjects": [
    {
      "organizationid": "ORGANIZATION NAME",
      "typemeter": "INAL",
      "timestamp": 1668603604000,
      "timestamp_to": 1668603604000,
      "lat": -1.0,
      "lon": -1.0,
      "cota": 0.0,
      "timezone": "Europe/Madrid",
      "description_origin": "Description data organization_topic",
      "origin": "inal.sf.1.35",
      "uid": "inal.sf.1.35",
      "alias": "inal.sf.1.35",
      "data": [
        {
          "name": "connections",
          "value": 1,
          "metric": "connections",
          "description": "number of connections"
        }
      ]
    }
  ]
}
```

Figura 25. Ejemplo llamada POST a la colección Kunna Gen Test. (Fuente propia)

Por lo tanto, se propuso una interfaz compleja y estructurada en varios pasos, que permitiría a los usuarios ir completando los distintos campos y valores necesarios para generar el JSON correspondiente y enviarlo a la API de Smart University.

Sin embargo, durante una reunión previa a la implementación, el tutor José Vicente sugirió un enfoque alternativo para simplificar la interfaz y el traspaso de información. Se propuso que el sistema actuara como emisor en el protocolo MQTT con Mosquitto, enviando el JSON generado, de modo que el receptor pudiera acceder a este JSON para su uso posterior en la visualización.

El cambio se fundamentó en las ventajas de utilizar MQTT, un protocolo ligero y fiable que está diseñado específicamente para aplicaciones IoT. Este enfoque reduce la complejidad del proceso al eliminar pasos intermedios, ya que Kunna podría conectarse directamente al tópico correspondiente y recibir las simulaciones en tiempo real. De esta manera, se optimiza la transferencia de datos, asegurando una mayor eficiencia y simplicidad en la arquitectura del sistema.

Con este nuevo enfoque, la interfaz se simplificará, incluyendo un campo para el nombre, otro para importar la colección de localizaciones deseadas y un campo de texto para introducir el JSON. El JSON seguirá un patrón específico que facilitará su extracción y generación, como se muestra en la Figura 33.

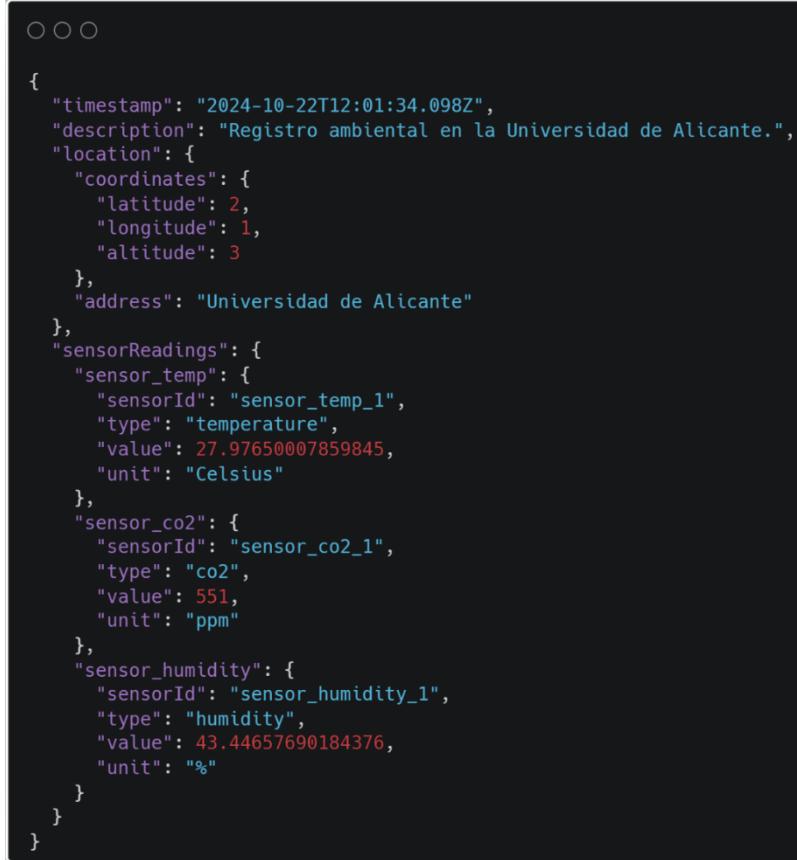
```
○ ○ ○  
{  
    "campo2": "^int[0,10]",  
    "campo3": "^float[20,25]",  
    "campo4": "^bool[8,9]",  
    "time": "^timenow",  
    "campo5": "este texto",  
    "campo6": "^array[4]int[0,50]",  
    "campo7": "^array[4]float[0,50]",  
    "campo8": "^array[4]bool",  
    "campo9": {  
        "campo10": "^array[4]float[0,50]",  
        "campo11": "^float[20,25]"  
    },  
    "campo12": "^positionlong",  
    "campo13": "^positionlat",  
    "campo14": "^positioncote"  
}
```

Figura 26. Patrón JSON para generar simulaciones. (Fuente propia)

Como se observa, cada campo está representado entre comillas, seguido del tipo de dato correspondiente. El uso del símbolo ^ indica que se trata de una instrucción para extraer la

información y determinar el tipo de dato. Tras la extracción, se implementará un método que, según la palabra clave (por ejemplo, ^int o ^array), generará el dato adecuado.

A continuación, se generará un nuevo JSON final que contendrá los datos generados. Por ejemplo:



```
{  
    "timestamp": "2024-10-22T12:01:34.098Z",  
    "description": "Registro ambiental en la Universidad de Alicante.",  
    "location": {  
        "coordinates": {  
            "latitude": 2,  
            "longitude": 1,  
            "altitude": 3  
        },  
        "address": "Universidad de Alicante"  
    },  
    "sensorReadings": {  
        "sensor_temp": {  
            "sensorId": "sensor_temp_1",  
            "type": "temperature",  
            "value": 27.97650007859845,  
            "unit": "Celsius"  
        },  
        "sensor_co2": {  
            "sensorId": "sensor_co2_1",  
            "type": "co2",  
            "value": 551,  
            "unit": "ppm"  
        },  
        "sensor_humidity": {  
            "sensorId": "sensor_humidity_1",  
            "type": "humidity",  
            "value": 43.44657690184376,  
            "unit": "%"  
        }  
    }  
}
```

Figura 27. Ejemplo JSON final con datos generados. (Fuente propia)

De esta manera, no solo se simplifica la interfaz, sino que también se garantiza una comunicación eficiente y estructurada a través del protocolo MQTT. Al seguir un patrón predefinido, se minimiza el riesgo de errores y se optimiza la interoperabilidad entre diferentes componentes del sistema.

Una vez definido el método de generación de las simulaciones, es fundamental plantear cómo realizaremos las llamadas CRUD necesarias para crear y gestionar estas simulaciones. Cada simulación constará de tres campos: un nombre, el ID de la localización seleccionada y los parámetros en formato JSON. Además, se establecerá una clave foránea que refiera al usuario, permitiendo asociar cada simulación con el creador correspondiente.

Es importante destacar que los usuarios básicos tendrán acceso restringido, pudiendo ver y manipular únicamente las simulaciones que ellos mismos hayan creado, mientras que los administradores podrán acceder a todas las simulaciones.

En cuanto a la interfaz, se propone que sea similar a las existentes, con una página inicial que muestre un listado de simulaciones, acompañada de botones para editar, eliminar, crear nueva y ejecutar simulación. También se incluirá un buscador para facilitar la localización de simulaciones. La página de creación contará con un formulario para introducir el nombre de la simulación, seleccionar la colección de localizaciones y un campo de texto para los parámetros.

#### 6.6.2. Fase de implementación

Una vez definidos los requerimientos, se inició la implementación, comenzando por el backend con la creación de un modelo y un controlador para las simulaciones. El modelo incluirá un campo de tipo string para el nombre de la colección, un objeto JSON que almacenará los parámetros, una clave foránea denominada locationId, que hará referencia a la localización asociada a la simulación, y otra clave foránea denominada userId, que hará referencia al usuario que ha creado la simulación. Esta relación se establece utilizando `Simulation.belongsTo(Locations, { foreignKey: 'locationId' })` y `Simulation.belongsTo(User, { foreignKey: 'userId' })`.

En cuanto al controlador, se desarrollarán los siguientes métodos:

- `newSimulation()`: Este método se encargará de crear una nueva simulación. Recibirá en el cuerpo de la solicitud el nombre de la colección, el id de la localización y el JSON con los parámetros, además de recuperar el ID del usuario desde el token de autenticación.
- `deleteSimulation()`: Este método recibirá un ID como parámetro y, si la simulación existe, procederá a eliminarla de la lista.
- `getAllSimulations()`: Este método obtendrá el rol y el ID del usuario. Si el usuario es un administrador, se devolverán todas las simulaciones. En caso contrario, si el usuario tiene un rol básico, se filtrarán las simulaciones por su id, devolviendo únicamente aquellas que ha creado.
- `getSimulationById()`: Este método recibirá un ID como parámetro y devolverá la simulación correspondiente a ese ID.
- `updateSimulation()`: Este método recibe un ID como parámetro y, en el cuerpo de la solicitud, los nuevos valores para la simulación. Luego, actualiza la simulación correspondiente reemplazando sus valores actuales con los nuevos proporcionados.

Una vez realizadas estas implementaciones, se procede a la creación del componente y sus interfaces, denominado Simulaciones. La página inicial (Figura 28) contendrá una tabla con cuatro filas: las tres primeras mostrarán la información de cada localización (ID, nombre, localización), y la última incluirá dos botones: uno para arrancar la simulación y otro con tres puntos que abre un desplegable con otros dos botones: uno para editar, que redirige al

componente correspondiente, y otro para eliminar, que al ser pulsado lanzará una llamada a deleteSimulation del servicio. También se incluirá un buscador para localizar una simulación específica en el listado, así como un botón de "Crear", que llevará al usuario a la página de crear simulación.

ID	Nombre	Localización
1	Simulación MQTT	Sensores CO2
2	S2	Aulas informática
6	ssssss	Sensores CO2

Figura 28. Página de gestión de simulaciones de la aplicación. (Fuente propia)

Los componentes de edición y creación compartirán el mismo formulario. En el caso de la creación, el formulario estará vacío y llamará a la función newSimulation del servicio, mientras que, en la edición, el formulario se completará con los datos correspondientes y llamará a la función updateSimulation, pasándole el ID de la simulación.

Como se puede observar en la Figura 36, el formulario incluye un campo para el nombre, otro desplegable para elegir la localización y, por último, un campo de texto para introducir el JSON con los parámetros. Al lado del campo de parámetros, un botón de información permite visualizar un ejemplo del formato requerido para el JSON. Al final del formulario, se presentan dos botones: uno para crear la simulación y otro para generar una vista previa de la simulación. Al hacer clic en “Generar Simulación”, la simulación de prueba se muestra en el panel derecho, lo que permite verificar que funciona correctamente antes de confirmar la creación. Cabe mencionar que hasta que no se genere la simulación no se podrá crear.

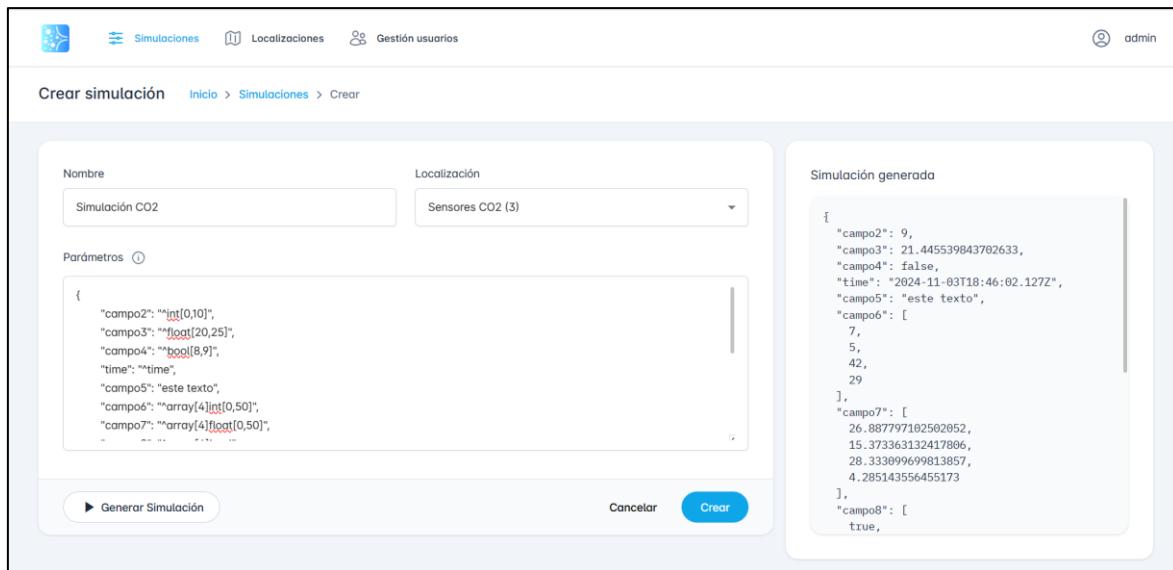


Figura 29. Página de crear simulación de la aplicación. (Fuente propia)

Para generar la simulación, es necesario almacenar un JSON proporcionado como plantilla y convertirlo en el JSON final. Como se detalló en el análisis, usaremos el símbolo ^ para indicar que un campo debe simularse. Si el campo no contiene este símbolo, se mantendrá el valor tal como se ingresó. Para implementar esta funcionalidad, creamos un método en el servicio de simulaciones llamado `generateNewJson()`, al cual se le pasan el JSON de parámetros, el ID de la ubicación seleccionada y un índice para identificar la posición de la ubicación que se va a usar.

El método `generateNewJson` (Figura 30) construye un nuevo JSON basado en los parámetros iniciales, asignando valores específicos en función de las instrucciones de simulación definidas en cada clave de `params`. En primer lugar, obtiene las coordenadas de la ubicación utilizando su ID. Luego, para cada clave en `params`, revisa si el valor tiene una instrucción especial:

- Genera números aleatorios enteros o decimales si el valor se marca como ^int o ^float, utilizando el rango especificado.
- Genera valores booleanos aleatorios si el valor está marcado como ^bool.
- Crea arrays personalizados, configurados con el tipo de datos y tamaño indicados, si el valor es ^array.
- Inserta la fecha actual en formato ISO si el valor es ^time.
- Asigna coordenadas específicas de la localización seleccionada (longitud, latitud, altura, alias) si el valor es ^position.

Si una clave en params contiene un objeto anidado, el método lo procesa de manera recursiva. El resultado es un JSON final que integra los datos simulados y las coordenadas de acuerdo con las especificaciones indicadas. Este JSON generado se guarda y se muestra en la simulación generada.

```

    ○○○

    // Método para generar un nuevo JSON basado en los parámetros-
    generateNewJson(params: any, locationId: number, numArrayLocalizaciones: number, time: Date): any {
        const newJson: any = {};

        // Llamar a getCoordinatesById y suscribirse para obtener las coordenadas-
        this._locationsService.getCoordinatesById(locationId, numArrayLocalizaciones).subscribe(coord => {
            // Una vez que tenemos las coordenadas, llenamos el newJson-
            for (const key in params) {
                if (params.hasOwnProperty(key)) {
                    const value = params[key];

                    if (typeof value === 'string' && value.startsWith('^')) {
                        if (value.startsWith('^int[')) {
                            const range = value.match(/[\(\d+),(\d+)\)]/);
                            if (range) {
                                const min = parseInt(range[1]);
                                const max = parseInt(range[2]);
                                newJson[key] = this.getRandomInt(min, max);
                            }
                        } else if (value.startsWith('^float[')) {
                            const range = value.match(/[\(\d+),(\d+)\)]/);
                            if (range) {
                                const min = parseFloat(range[1]);
                                const max = parseFloat(range[2]);
                                newJson[key] = this.getRandomFloat(min, max);
                            }
                        } else if (value.startsWith('^bool')) {
                            newJson[key] = Math.random() < 0.5; // true o false aleatorio-
                        } else if (value.startsWith('^array')) {
                            // Ajustar la expresión regular para capturar tanto el tamaño como el tipo de elemento-
                            const arrayDetails = value.match(/^\^array\[(\d+)\](\w+)(?:\[(\d+)\],(\d+)\])?/);

                            if (arrayDetails) {
                                const arrayLength = parseInt(arrayDetails[1]); // Longitud del array-
                                const elementType = arrayDetails[2]; // Tipo de elemento: 'int', 'float', 'bool'-
                                //
                                // Inicializa min y max solo si el tipo es int o float-
                                let min = 0;
                                let max = 100;

                                // Comprueba si es necesario obtener min y max-
                                if (arrayDetails[3] && arrayDetails[4]) {
                                    min = parseInt(arrayDetails[3]);
                                    max = parseInt(arrayDetails[4]);
                                }

                                // Generar el array dependiendo del tipo-
                                if (elementType === 'bool') {
                                    newJson[key] = this.generateBooleanArray(arrayLength);
                                } else {
                                    newJson[key] = this.generateArray(arrayLength, elementType, min, max);
                                }
                            }
                        } else if (value === '^time') {
                            newJson[key] = time.toISOString();
                        } else if (value.startsWith('^positionlong')) {
                            newJson[key] = coord.long; // Aquí asignamos la longitud-
                        } else if (value.startsWith('^positionlat')) {
                            newJson[key] = coord.lat; // Aquí asignamos la latitud-
                        } else if (value.startsWith('^positioncote')) {
                            newJson[key] = coord.height; // Aquí asignamos la altura-
                        } else if (value.startsWith('^positionalias')) {
                            newJson[key] = coord.alias; // Aquí asignamos la altura-
                        }
                    } else if (typeof value === 'object' && !Array.isArray(value)) {
                        newJson[key] = this.generateNewJson(value, locationId, numArrayLocalizaciones, time);
                    } else {
                        newJson[key] = value; // Copiar otros valores como están-
                    }
                }
            }
        });
    }

    return newJson; // Aquí regresamos el nuevo JSON-
}

```

Figura 30. Método `generateNewJson()` encargado de generar el JSON final de la simulación. (Fuente propia)

Una vez creados los componentes y comprobado su funcionamiento, el siguiente paso fue gestionar el acceso según el rol de cada usuario. Como mencionamos anteriormente, los usuarios básicos solo podrán ver y manipular las simulaciones que ellos mismos hayan creado, mientras que los administradores tendrán acceso a todas las simulaciones. No fue necesario realizar cambios adicionales en este aspecto, ya que el método getAllSimulations estaba diseñado para filtrar las localizaciones en función del rol: si el usuario es administrador, se devuelven todas; si es básico, se filtran por su ID, mostrando solo las suyas.

Además, al igual que en otras rutas del sistema, se implementó un AuthGuard para restringir el acceso a usuarios no autenticados, aplicando la propiedad canActivate: [AuthGuard] en las rutas correspondientes.

## 6.7. Iteración 7. Expansión de parámetros y nuevas funcionalidades de simulación

En esta iteración, el objetivo principal fue mejorar las simulaciones mediante la incorporación de nuevos parámetros y funcionalidades. Tras generar una simulación única a modo de prueba, el siguiente paso consistió en añadir la posibilidad de configurar el patrón de simulación. Esto incluye establecer el número de elementos a simular, la frecuencia temporal de las simulaciones y la cantidad de simulaciones por instante. Además, se consideró relevante permitir al usuario definir una fecha de inicio para las simulaciones.

Otro aspecto destacado fue la incorporación de dos modalidades de ejecución: una simulación en tiempo real, que permite observar el progreso en la interfaz y pausarla o detenerla según sea necesario, y una ejecución instantánea que simula todos los eventos de forma inmediata. Como en fases anteriores, esta iteración comenzó con un análisis detallado de los requisitos antes de proceder a la implementación.

### 6.7.1. Fase de análisis

Esta iteración surgió a partir de una reunión con el tutor en la que se discutieron los próximos pasos a seguir. El punto de partida fue la capacidad de crear una simulación única a modo de prueba, en la que se seleccionaba una ubicación aleatoria del sensor y, a partir de los parámetros definidos, se generaba la simulación. Este enfoque permitió verificar que los parámetros introducidos eran correctos y que los resultados obtenidos eran los esperados.

A partir de esta base, se identificó la necesidad de ampliar las funcionalidades para definir un comportamiento más complejo en las simulaciones. Esto incluye la capacidad de establecer cuántas simulaciones se desean generar (incluyendo la opción de un número infinito), cuántas simulaciones se ejecutan por instante, y si las ubicaciones de los sensores pueden repetirse o no. También se añadió la posibilidad de definir el intervalo de tiempo entre simulaciones, así como la fecha de inicio, lo que permite realizar simulaciones tanto de eventos pasados como futuros.

Para integrar estas nuevas capacidades, se propuso una interfaz renovada que incluye dos modalidades principales. La primera es una simulación en tiempo real, donde los usuarios pueden observar las simulaciones generándose en tiempo real, con opciones para pausar o detener el proceso. Esto resulta útil en situaciones donde se requiere monitorear el progreso de los datos y tomar decisiones dinámicas durante el proceso de simulación. La segunda modalidad permite ejecutar toda la simulación de forma instantánea, siendo útil especialmente en escenarios de prueba o en situaciones donde no se requiere simular un proceso en tiempo real, sino más bien generar los datos de forma inmediata y sin demoras. Estas funciones se implementarán tanto en el panel de detalle de cada simulación como en un listado general, lo que facilita iniciar varias simulaciones a la vez o profundizar en una específica de manera más detallada.

Finalmente, se planteó un nuevo enfoque para la interfaz de las simulaciones. En lugar de separar las opciones de creación y edición en pantallas diferentes, se optó por un sistema más dinámico con un botón de guardar. Este cambio permite a los usuarios modificar los parámetros de una simulación en cualquier momento, guardar los cambios y probar las nuevas configuraciones directamente desde la misma pantalla, optimizando así el flujo de trabajo y la experiencia de uso.

Por otro lado, aunque no está directamente relacionado con el objetivo principal de la iteración, se decidió renombrar las "localizaciones" como "sensores". Este ajuste responde a que el término "sensores" refleja mejor su propósito y función, ya que son los encargados de recopilar los datos para las simulaciones, alineándose mejor con el enfoque general del sistema. Además, habrá que incorporar tres nuevos campos para los sensores: dev\_eui, join\_eui y dev\_addr, que representan datos específicos del sensor, necesarios para identificarlos.

#### 6.7.2. Fase de implementación

Una vez definidos los requerimientos, se procedió con la implementación, comenzando por la adición de los nuevos campos a la simulación. Los campos añadidos son los siguientes:

- minRegistrosPorInstante: define el número mínimo de registros que se generarán por instante.

- maxRegistrosPorInstante: define el número máximo de registros por instante.
- minIntervaloEntreRegistros: establece el intervalo mínimo de tiempo entre los registros generados.
- maxIntervaloEntreRegistros: establece el intervalo máximo entre los registros generados.
- numElementosASimular: determina el número total de elementos que se simularán (0 significa infinito).
- noRepetirCheckbox: indica si se deben evitar las repeticiones en las localizaciones de los sensores (0 para repetir elementos y 1 para no repetir elementos).
- date: establece la fecha de inicio de la simulación.

Estos campos fueron añadidos al modelo de simulaciones en el backend, y se realizaron ajustes en el controlador para modificar los métodos relacionados. También se actualizó la interfaz y el formulario para incluir estos nuevos campos. Como se muestra en la Figura 38, los nuevos campos se integraron en el formulario, destacando especialmente el campo de "No repetir elementos". Al activar esta opción, si el número máximo de registros por instante excede el número de localizaciones disponibles en el sensor, se mostrará un mensaje de error impidiendo la acción. Además, debajo del campo de la fecha de inicio, se añadió un botón de "Usar fecha actual", que, al seleccionarse, establece automáticamente la fecha al valor actual.

The screenshot shows the 'Simulaciones' (Simulations) section of the application. On the left, there are input fields for 'Nombre' (Name), 'Sensores' (Sensors), 'Nº registros en cada instante' (Number of records per instant), 'Tiempo entre registros (seg.)' (Time between records (sec.)), 'Total registros a simular (0 significa infinito)' (Total number of records to simulate (0 means infinite)), and 'Fecha de Inicio (DD/MM/YYYY HH:mm:ss)' (Start date (DD/MM/YYYY HH:mm:ss)). A checkbox labeled 'No repetir elementos' (Do not repeat elements) is checked. Below these, there is a 'Parámetros' (Parameters) section containing a JSON configuration:

```
{
  "campo2": "int[0,10]",
  "campo3": "float[20,25]",
  "campo4": "bool[8,9]",
  "time": "time",
  "campo5": "este texto",
  "campo6": "arrav4lnt[0,50]"
}
```

On the right, a large text area titled 'Simulación' (Simulation) displays the generated JSON data:

```
{
  "campo2": 1,
  "campo3": 20.9607653255437,
  "campo4": true,
  "time": "2024-11-12T11:06:00.000Z",
  "campo5": "este texto",
  "campo6": [
    37,
    0,
    39,
    22
  ],
  "campo7": [
    11.271765899517238,
    44.631284239327775,
    33.013221289907655,
    2.2179619217002955
  ],
  "campo8": [
    false,
    true,
    false,
    false
  ],
  "campo9": [
    "2024-11-12T11:06:00.000Z"
  ]
}
```

At the bottom, there are buttons for 'Probar Simulación' (Test Simulation), 'Cancelar' (Cancel), and 'Guardar' (Save).

Figura 31. Página de crear simulación de la aplicación con los nuevos campos. (Fuente propia)

Por lo tanto, ahora a la hora de generar cada simulación, se partirá de la fecha seleccionada en lugar del momento actual como ocurría antes.

El siguiente paso es realizar los métodos para simular en vivo o simular instantáneamente con los parámetros que acabamos de definir.

Empezaremos por el método `simularInstantaneamente()` (Figura 32), que permite ejecutar simulaciones de manera continua sin esperar intervalos entre pasos. Comienza obteniendo los parámetros de simulación mediante el `simulationId` y prepara los datos iniciales, como el número total de elementos generados y el tiempo inicial de la simulación. En cada paso, calcula un número aleatorio de registros a generar dentro del rango definido por los parámetros `minRegistrosPorInstante` y `maxRegistrosPorInstante`. Si está habilitada la opción de no repetir localizaciones (`noRepetirCheckbox`), se seleccionan índices únicos de entre las coordenadas disponibles del sensor; en caso contrario, los índices se eligen aleatoriamente. Los parámetros de la simulación se procesan y se generan nuevos datos en formato JSON, que son enviados a través de MQTT y almacenados para seguimiento.

El proceso continúa iterativamente hasta alcanzar el número máximo de elementos a simular, avanzando el tiempo simulado para cada paso según un intervalo aleatorio definido entre `minIntervaloEntreRegistros` y `maxIntervaloEntreRegistros`. Al finalizar, se detiene la simulación y devuelve los resultados generados a través de un callback.

```

    ○○○

    // Método para iniciar la simulación sin esperar entre intervalos
    simularInstantaneamente(simulationId: number, callback: (result: any) => void): void {
        this.totalGenerados[simulationId] = 0;
        this.simulacionesGeneradas = [];

        this.getSimulationById(simulationId).subscribe((simulacion) => {
            let totalGenerados = 0;
            let time = new Date(simulacion.date);

            const executeSimulationStep = () => {
                if (totalGenerados >= simulacion.numElementosASimular) {
                    this.isRunning[simulationId] = false;
                    callback(this.simulacionesGeneradas);
                    return;
                }

                let registrosEnEsteInstante = Math.floor(
                    Math.random() * (simulacion.maxRegistrosPorInstante - simulacion.minRegistrosPorInstante + 1)
                ) + simulacion.minRegistrosPorInstante;

                if (totalGenerados + registrosEnEsteInstante > simulacion.numElementosASimular) {
                    registrosEnEsteInstante = simulacion.numElementosASimular - totalGenerados;
                }

                let usedIndices: Set<number> = new Set();
                for (let j = 0; j < registrosEnEsteInstante; j++) {
                    const currentRecordTime = new Date(time);

                    this._sensoresService.getSensorById(simulacion.sensorId).subscribe((sensor) => {
                        let randomIndex: number;
                        if (simulacion.noRepetirCheckbox === 1) {
                            const availableIndices = sensor.coordinates
                                .map((_, idx) => idx)
                                .filter((idx) => !usedIndices.has(idx));
                            randomIndex = availableIndices[Math.floor(Math.random() * availableIndices.length)];
                            usedIndices.add(randomIndex);
                        } else {
                            randomIndex = Math.floor(Math.random() * sensor.coordinates.length);
                        }

                        let parameters = typeof simulacion.parameters === "string"
                            ? JSON.parse(simulacion.parameters)
                            : simulacion.parameters;

                        this.generateNewJson(parameters, simulacion.sensorId, randomIndex, currentRecordTime).then((newSimulacion) =>
{
                            this.simulacionesGeneradas.push(newSimulacion);
                            this.sendMessageMqtt(newSimulacion);
                        });
                    });
                }
            });

            totalGenerados += registrosEnEsteInstante;
            this.totalGenerados[simulationId] = totalGenerados;

            const intervalo = Math.floor(
                Math.random() * (simulacion.maxIntervaloEntreRegistros - simulacion.minIntervaloEntreRegistros + 1)
            ) + simulacion.minIntervaloEntreRegistros;
            time = new Date(time.getTime() + intervalo * 1000);

            executeSimulationStep();
        });
    }

    executeSimulationStep();
}

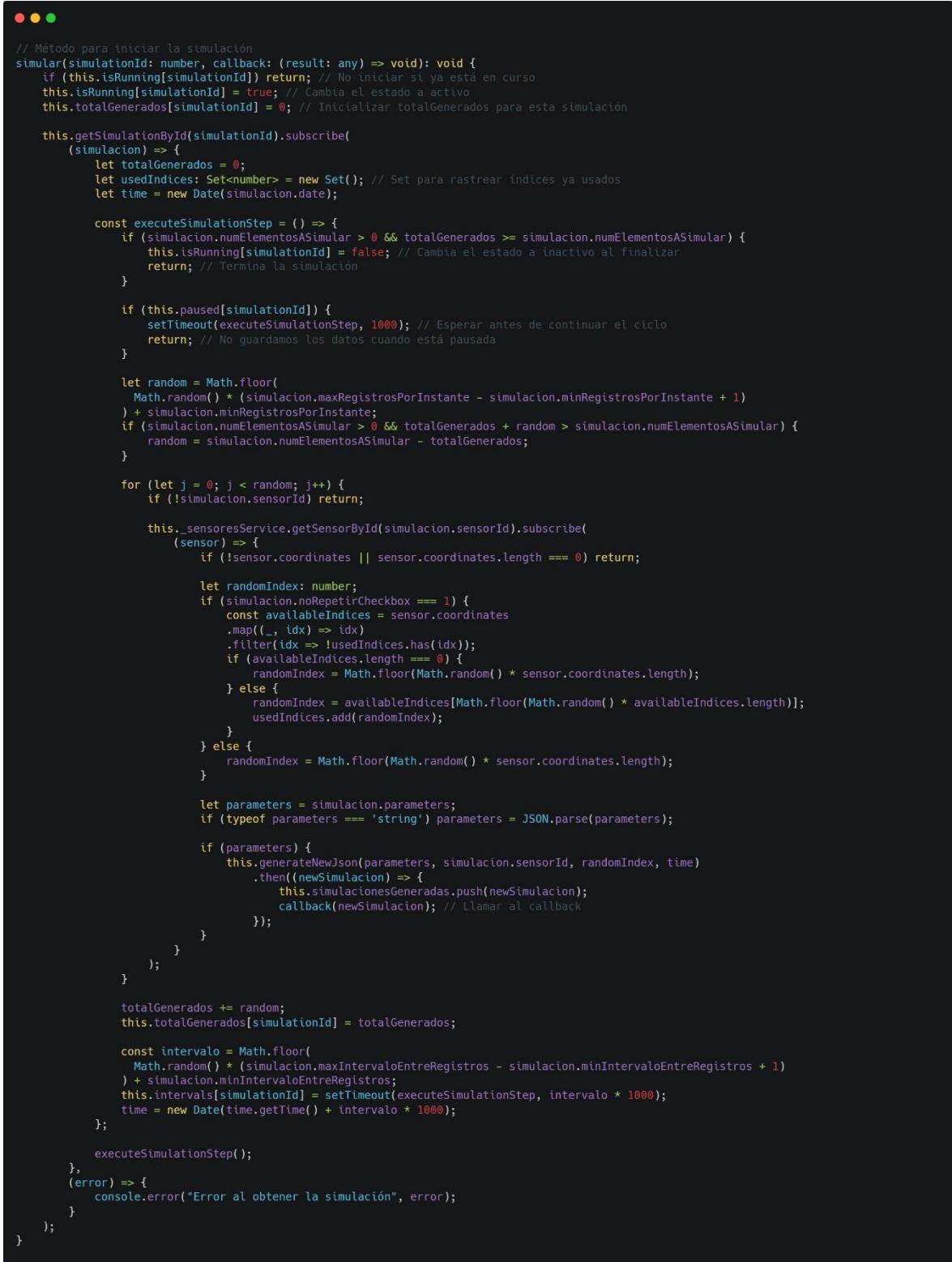
```

*Figura 32. Método simularInstantaneamente() encargado de generar simulaciones sin intervalo de tiempo.  
(Fuente propia)*

Por otro lado, el método simular() (Figura 33) nos permite ejecutar simulaciones de manera controlada, respetando un intervalo de tiempo entre cada paso. El proceso es similar al de simularInstantaneamente(), pero incluye la pausa entre intervalos. Comienza verificando que la simulación no esté en curso y establece su estado como activo. Luego, obtiene los parámetros de simulación mediante el simulationId y prepara los datos iniciales, como el tiempo de inicio y el contador de elementos generados.

En cada iteración, calcula un número aleatorio de registros a generar dentro del rango definido por `minRegistrosPorInstante` y `maxRegistrosPorInstante`. Si está habilitada la opción de no repetir localizaciones (`noRepetirCheckbox`), se seleccionan índices únicos de las coordenadas disponibles del sensor; en caso contrario, los índices se eligen aleatoriamente. Los parámetros de simulación se procesan y se generan nuevos datos en formato JSON, que son enviados por MQTT, almacenados y devueltos a través de un callback.

El proceso se repite iterativamente después de un intervalo aleatorio definido entre `minIntervaloEntreRegistros` y `maxIntervaloEntreRegistros`, actualizando el tiempo simulado y el contador de elementos generados. La simulación continúa hasta alcanzar el número máximo de registros configurados o indefinidamente si no se establece un límite. Una vez completada, el método detiene la simulación y registra los resultados finales.



```

// Método para iniciar la simulación
simular(simulationId: number, callback: (result: any) => void): void {
    if (this.isRunning[simulationId]) return; // No iniciar si ya está en curso
    this.isRunning[simulationId] = true; // Cambia el estado a activo
    this.totalGenerados[simulationId] = 0; // Inicializar totalGenerados para esta simulación

    this.getSimulationById(simulationId).subscribe(
        (simulacion) => {
            let totalGenerados = 0;
            let usedIndices: Set<number> = new Set(); // Set para rastrear índices ya usados
            let time = new Date(simulacion.date);

            const executeSimulationStep = () => {
                if (simulacion.numElementosASimular > 0 && totalGenerados >= simulacion.numElementosASimular) {
                    this.isRunning[simulationId] = false; // Cambia el estado a inactivo al finalizar
                    return; // Termina la simulación
                }

                if (this.paused[simulationId]) {
                    setTimeout(executeSimulationStep, 1000); // Esperar antes de continuar el ciclo
                    return; // No guardamos los datos cuando está pausada
                }

                let random = Math.floor(
                    Math.random() * (simulacion.maxRegistrosPorInstante - simulacion.minRegistrosPorInstante + 1)
                ) + simulacion.minRegistrosPorInstante;
                if (simulacion.numElementosASimular > 0 && totalGenerados + random > simulacion.numElementosASimular) {
                    random = simulacion.numElementosASimular - totalGenerados;
                }

                for (let j = 0; j < random; j++) {
                    if (!simulacion.sensorId) return;

                    this._sensoresService.getSensorById(simulacion.sensorId).subscribe(
                        (sensor) => {
                            if (!sensor.coordinates || sensor.coordinates.length === 0) return;

                            let randomIndex: number;
                            if (simulacion.noRepetirCheckbox === 1) {
                                const availableIndices = sensor.coordinates
                                    .map((_, idx) => idx)
                                    .filter(idx => !usedIndices.has(idx));
                                if (availableIndices.length === 0) {
                                    randomIndex = Math.floor(Math.random() * sensor.coordinates.length);
                                } else {
                                    randomIndex = availableIndices[Math.floor(Math.random() * availableIndices.length)];
                                    usedIndices.add(randomIndex);
                                }
                            } else {
                                randomIndex = Math.floor(Math.random() * sensor.coordinates.length);
                            }

                            let parameters = simulacion.parameters;
                            if (typeof parameters === 'string') parameters = JSON.parse(parameters);

                            if (parameters) {
                                this.generateNewJson(parameters, simulacion.sensorId, randomIndex, time)
                                    .then((newSimulacion) => {
                                        this.simulacionesGeneradas.push(newSimulacion);
                                        callback(newSimulacion); // Llamar al callback
                                    });
                            }
                        }
                    );
                }

                totalGenerados += random;
                this.totalGenerados[simulationId] = totalGenerados;

                const intervalo = Math.floor(
                    Math.random() * (simulacion.maxIntervaloEntreRegistros - simulacion.minIntervaloEntreRegistros + 1)
                ) + simulacion.minIntervaloEntreRegistros;
                this.intervals[simulationId] = setTimeout(executeSimulationStep, intervalo * 1000);
                time = new Date(time.getTime() + intervalo * 1000);
            };
            executeSimulationStep();
        },
        (error) => {
            console.error("Error al obtener la simulación", error);
        }
    );
}
}

```

Figura 33. Método `simular()` encargado de generar simulaciones en tiempo real. (Fuente propia)

Una vez definidos los métodos y comprobado su correcto funcionamiento, procedemos a crear las interfaces necesarias para reflejar las simulaciones. En la página del listado de simulaciones (Figura

35), hemos añadido dos botones: un ícono de cohete, que genera la simulación instantáneamente, y un triángulo de "play", que inicia la simulación de manera progresiva.



*Figura 34. Botones para generar simulaciones de manera instantánea o progresiva. (Fuente propia).*

Al hacer clic en el ícono de cohete, se invoca el método `simularInstantaneamente()` del servicio, pasándole el ID de la simulación seleccionada. Este método genera la simulación de forma inmediata y notifica al usuario del resultado. Por otro lado, al presionar el botón de "play", se llama al método `simular()` del servicio, enviándole el ID de la simulación seleccionada, lo que permite ejecutar la simulación en tiempo real.

Para visualizar este progreso de forma dinámica, se ha implementado un array en el componente de simulaciones que almacena las simulaciones activas. Estas simulaciones se muestran en una nueva pantalla desplegable donde el usuario puede observar su avance en tiempo real. Además, es posible ejecutar múltiples simulaciones simultáneamente, con opciones para pausarlas o detenerlas según sea necesario, lo que facilita un manejo rápido y eficiente de varias simulaciones al mismo tiempo.

Es importante destacar que, en los casos donde el número de elementos a generar sea ilimitado, no será posible utilizar la opción de simulación instantánea.

*Figura 35. . Página de listado de simulaciones de la aplicación con los nuevos botones y funcionalidades.  
(Fuente propia)*

Por otro lado, se ha desarrollado una página dedicada para gestionar cada simulación de manera individual. En esta página, los usuarios pueden modificar los parámetros de la simulación, guardar los cambios realizados y ejecutar las simulaciones correspondientes. Esto nos permite ir realizando cambios y pruebas sobre una simulación e ir observando el progreso en tiempo real dentro de la misma interfaz.

En esta página, se utiliza el mismo formulario empleado para crear una simulación, con la diferencia de que, una vez creada, en el panel de la derecha podemos gestionar la simulación. Este panel incluye dos botones, un ícono de cohete y otro de "play", similares a los de la página de listado.

Al hacer clic en el ícono del cohete, se invoca el método `simularInstantáneamente()` del servicio, generando la simulación completa. Los resultados se muestran directamente en el panel, donde el usuario puede clicar en cualquiera de ellos y ampliarlos para analizarlos en detalle.

Nombre: Simulación de prueba  
Sensores: Sensores C02 (6)  
Nº registros en cada instante: 4, 6 (checked "No repetir elementos"), Tiempo entre registros (seg.): 15, 20  
Total registros a simular (0 significa infinito): 164  
Fecha de inicio (DD/MM/YYYY HH:mm:ss): 14/11/2024 12:00:00  
User fecha actual:  
Parámetros:

```
{
  "campo2": "int[0,10]",
  "campo3": "float[20,25]",
  "campo4": "bool[8,9]",
  "time": "time",
  "campo5": "este texto",
  "campo6": "array[4]float[0,50]",
  "campo7": "array[4]float[0,50]"
}
```

Simulación: 0: (campo2: 6, campo3: 23.775831192254107, campo4: true, ...)

Figura 36. Simulación generada instantáneamente a partir de una simulación de prueba. (Fuente propia)

Por otro lado, al hacer clic en el botón de “play”, se invoca el método `simular()` del servicio, iniciando la simulación en tiempo real. En el panel de la derecha, se visualiza cómo se van generando las simulaciones, mientras que en la parte inferior aparece una barra de progreso que muestra el número de elementos generados. Además, se disponen de dos botones para pausar o detener la simulación en cualquier momento.

Nombre: Simulación de prueba  
Sensores: Sensores C02 (6)  
Nº registros en cada instante: 4, 6 (checked "No repetir elementos"), Tiempo entre registros (seg.): 15, 20  
Total registros a simular (0 significa infinito): 164  
Fecha de inicio (DD/MM/YYYY HH:mm:ss): 14/11/2024 12:00:00  
User fecha actual:  
Parámetros:

```
{
  "campo2": "int[0,10]",
  "campo3": "float[20,25]",
  "campo4": "bool[8,9]",
  "time": "time",
  "campo5": "este texto",
  "campo6": "array[4]float[0,50]",
  "campo7": "array[4]float[0,50]"
}
```

Simulación: 0: (campo2: 9, campo3: 20.51954008462239, campo4: false, ...)

Elementos generados: 15 / 164

Figura 37. Simulación en tiempo real a partir de una simulación de prueba. (Fuente propia)

Una vez que la interfaz está completamente funcional, generando simulaciones y reflejando todo el progreso, el siguiente paso es configurar MQTT para la emisión de mensajes. Esto permitirá enviar los datos generados al servidor de Kunna, donde serán procesados y visualizados correctamente, lo cual dejaremos para la siguiente iteración.

## 6.8. Iteración 8. Integración y emisión de mensajes con MQTT

En esta iteración, el objetivo principal fue integrar MQTT para la emisión de mensajes desde nuestra aplicación al software de Kunna. Esta integración nos permitirá enviar las simulaciones generadas, para que sean procesadas y visualizadas correctamente en el sistema de Kunna. Más concretamente, el protocolo MQTT sigue una arquitectura de publicación/suscripción, en la cual nuestra aplicación actuará como publicador, enviando los mensajes, y Kunna se suscribirá para recibirlas y procesarlos en tiempo real.

La iteración comenzará con una fase de análisis en la que se estudiará el protocolo MQTT y su funcionamiento, para luego pasar a su implementación en el sistema.

### 6.8.1. Fase de análisis

Al ser una tecnología nueva en el desarrollo, el primer paso fue investigar a fondo el protocolo y comprender su funcionamiento. MQTT [14] es un protocolo de mensajería estándar de OASIS diseñado para el Internet de las Cosas (IoT). Se basa en un modelo de publicación/suscripción extremadamente ligero, lo que lo hace ideal para conectar dispositivos remotos con una huella de código mínima y un bajo consumo de ancho de banda. Actualmente, MQTT se utiliza en diversas industrias, como la automotriz, manufacturera, telecomunicaciones, petróleo y gas, entre otras.

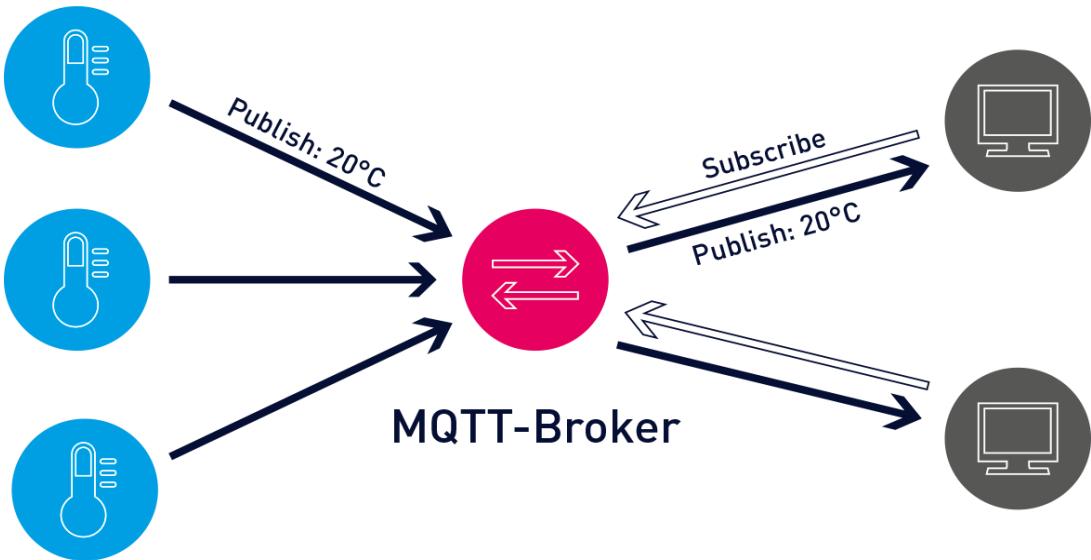


Figura 38. Protocolo MQTT. (<https://www.paessler.com/es/it-explained/mqtt>)

MQTT se ejecuta sobre TCP/IP utilizando una topología PUSH/SUBSCRIBE. En esta arquitectura, existen dos tipos de sistemas: clientes y brókeres. Un bróker es el servidor central con el que se comunican los clientes, recibiendo los mensajes de unos y enviándolos a otros. Los clientes no se comunican directamente entre sí, sino que se conectan a través del bróker. Cada cliente puede actuar como publicador, suscriptor o incluso ambos, dependiendo de sus necesidades.

El protocolo es controlado por eventos, lo que significa que no hay transmisión de datos periódica o continua, sino que los mensajes se envían solo cuando hay información disponible para compartir. De esta forma, el volumen de transmisión se mantiene al mínimo. Un cliente publica información únicamente cuando hay nuevos datos que enviar, y el bróker distribuye esos datos a los suscriptores solo cuando llegan nuevos mensajes.

En el caso de nuestra aplicación, esta actuará como publicador, generando las simulaciones que luego serán enviadas al bróker mediante mensajes MQTT. Kunna, en su rol de suscriptor, se suscribirá a los canales relevantes para recibir estos mensajes en tiempo real. A medida que nuestra aplicación genera simulaciones, estas serán enviadas a través de MQTT, permitiendo a Kunna recibir y procesar los datos de manera inmediata.

Este flujo de información permite que Kunna visualice las simulaciones generadas en tiempo real, lo que facilita su procesamiento y análisis sin necesidad de intervenciones manuales. A través de este proceso, nuestra aplicación se integra de forma fluida con el sistema de Kunna, permitiendo la creación y monitoreo de simulaciones de manera eficiente y continua.

Una vez comprendido su funcionamiento, el siguiente paso fue investigar cómo integrar MQTT en nuestro sistema Angular. Para ello, lo primero será configurar una biblioteca MQTT como mqtt.js, la cual instalamos con el comando: `npm install mqtt --save`. A continuación, habrá que configurar el cliente creando un servicio MQTT que gestionará la conexión y el envío de mensajes. Este servicio se importará en nuestro componente para realizar las llamadas correspondientes.

Para probar la funcionalidad de MQTT antes de integrarlo con el sistema de Kunna, decidimos utilizar HiveMQ. HiveMQ [16] es una plataforma de mensajería MQTT altamente escalable y robusta, diseñada específicamente para aplicaciones IoT. Este servidor nos ofrece un entorno de pruebas para asegurar que la integración de MQTT funciona correctamente, garantizando que los mensajes se envíen y reciban sin problemas antes de pasar a la fase de producción, donde Kunna gestionará la comunicación mediante su propio broker.

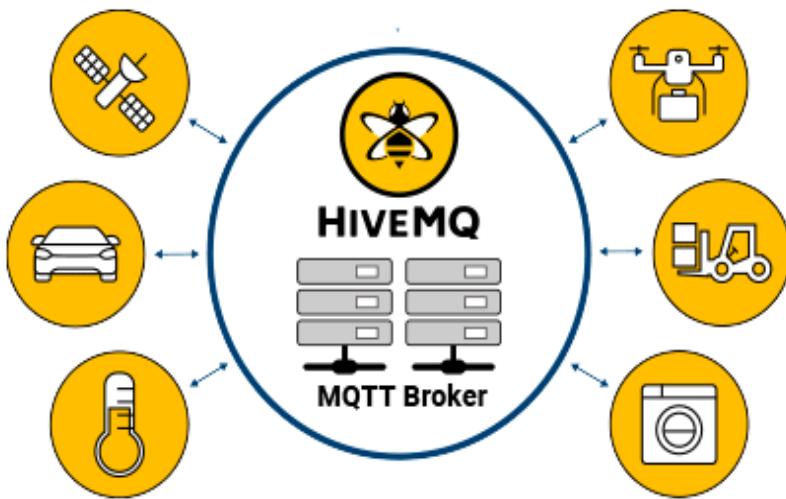


Figura 39. HiveMQ, servidor utilizado como entorno de pruebas de MQTT.  
(<https://www.foxon.cz/en/services/integrate-sw-platforms/hivemq-mqtt-broker>)

Una vez probado el correcto funcionamiento de la integración con HiveMQ, el siguiente paso será conectar nuestra aplicación al bróker final de Kunna y al tópico correspondiente. Dado que existe la posibilidad de querer enviar mensajes a diferentes brókers y tópicos, se planteó la necesidad de añadir una nueva funcionalidad al sistema. Esta funcionalidad permitirá crear y gestionar tópicos a los cuales se puedan emitir mensajes, y durante la simulación, se podrá seleccionar a cuál o cuáles de estos tópicos se desea enviar la información. De esta manera, se ofrece flexibilidad en la gestión de la emisión de mensajes, permitiendo que la simulación se pueda distribuir en múltiples destinos si fuera necesario.

#### 6.8.2. Fase de implementación

Una vez definidos los requerimientos, se procedió con la implementación, comenzando por instalar la librería de MQTT. Para ello, utilizamos el siguiente comando: `npm install mqtt --save`.

A continuación, crearemos un archivo para gestionar el servicio de MQTT, llamado `mqqt.service`. Este se encargará de gestionar la conexión y el envío de mensajes, y se importará en nuestro componente para realizar las llamadas correspondientes.

El primer paso fue definir el constructor, encargado de inicializar la conexión al broker MQTT. Se utilizó la función `mqtt.connect()` para establecer la comunicación con el broker a través de WebSockets seguros (TLS), especificando la URL del broker y el puerto correspondiente.

En la configuración, se incluyó un `clientId` único para identificar al cliente, junto con las credenciales de acceso (`username` y `password`) en caso de ser necesarias para autenticación. Además, se definieron parámetros como la opción `clean` para iniciar una sesión nueva, un período de reconexión automática (`reconnectPeriod`), y un tiempo de mantenimiento de la conexión (`keepalive`).

Finalmente, se configuraron manejadores de eventos como `connect` para confirmar la conexión exitosa, `error` para capturar fallos en la conexión, y `message` para procesar los mensajes recibidos en diferentes tópicos.

```
constructor() {
  // Configuración para conectarse a un broker MQTT usando WebSockets TLS
  this.client = mqtt.connect('wss://your-broker-url:port/mqtt', {
    clientId: 'your-client-id', // Cambia este ID, asegúrate de que sea único
    username: 'your-username', // Nombre de usuario (si no se usa autenticación, puedes dejarlo vacío)
    password: 'your-password', // Contraseña (si no se usa autenticación, puedes dejarlo vacío)
    clean: true, // Activa una sesión limpia para este cliente
    reconnectPeriod: 1000, // Intentará reconectar cada segundo si la conexión falla
    keepalive: 60, // Enviar un ping al broker cada 60 segundos
  });

  this.client.on('connect', () => {
    console.log('Conectado al broker MQTT');
  });

  this.client.on('error', (err) => {
    console.error('Error de conexión MQTT:', err);
  });

  this.client.on('message', (topic, message) => {
    // Aquí manejas los mensajes que recibes
    console.log('Mensaje recibido en el tópico', topic, ':', message.toString());
  });
}
```

Figura 40. Constructor encargado de establecer la conexión MQTT en `mqqt.service`. (Fuente propia)

Después de definir el constructor, se implementaron dos métodos esenciales para la interacción con el broker MQTT: sendMessage y subscribeToTopic. Estos métodos permiten enviar mensajes a tópicos específicos y suscribirse a ellos para recibir información, respectivamente.

El método sendMessage se encarga de publicar mensajes en un tópico del broker. Antes de enviar un mensaje, verifica si el cliente MQTT está conectado. Si la conexión está activa, utiliza publish para enviar el mensaje y maneja posibles errores, mostrando un mensaje de éxito o de error en la consola. Este enfoque garantiza que los mensajes solo se envíen cuando la conexión es estable.

Por otro lado, el método subscribeToTopic permite al cliente suscribirse a un tópico en particular. Mediante el método subscribe, se establece la suscripción y se gestionan errores potenciales durante el proceso. Si la suscripción es exitosa, se notifica en la consola, indicando que el cliente podrá recibir mensajes publicados en ese tópico. Ambos métodos son fundamentales para establecer una comunicación completa y funcional en un sistema basado en MQTT.

```
sendMessage(topic: string, message: string): void {
  if (this.client.connected) {
    this.client.publish(topic, message, (err) => {
      if (err) {
        console.error('Error al enviar mensaje:', err);
      } else {
        console.log('Mensaje enviado al tópico', topic, ':', message);
      }
    });
  } else {
    console.error('Cliente MQTT no conectado.');
  }
}

// Puedes suscribirte a un tópico
subscribeToTopic(topic: string): void {
  this.client.subscribe(topic, (err) => {
    if (err) {
      console.error('Error al suscribirse al tópico:', err);
    } else {
      console.log('Suscripción exitosa al tópico:', topic);
    }
  });
}
```

Figura 41. Métodos sendMessage() y subscribeToTopic de mqqt.service. (Fuente propia)

Lo siguiente que se hizo fue importar el servicio de MQTT en nuestro componente y crear un método llamado sendMessageMQTT(). Este método tiene la responsabilidad de enviar las simulaciones generadas al broker MQTT. Para ello, recibe el mensaje como parámetro, lo convierte a una cadena de texto (stringify) y lo publica utilizando el servicio MQTT previamente configurado, enviándolo al tópico simulaciones, inicialmente utilizado como un canal de prueba.

```
// Enviar mensaje MQTT
sendMessageMqtt(message: String) {
    const messageMQTT = JSON.stringify(message);
    this.mqttService.sendMessage('simulaciones', messageMQTT);
}
```

*Figura 42. Método sendMessageMqtt() encargado de enviar la simulación generada al servicio de MQTT. (Fuente propia)*

Ahora en el componente, cada vez que se genera un nuevo JSON con los datos de una simulación, se invoca al método sendMessageMqtt() para enviar el mensaje generado al bróker. Esto asegura una integración fluida entre la generación de datos y su publicación en el sistema.

```
// Generar el nuevo JSON con el tiempo específico para este registro
this.generateNewJson(parameters, simulacion.sensorId, randomIndex, currentRecordTime)
.then((newSimulacion) => {
    this.simulacionesGeneradas.push(newSimulacion);
    this.sendMessageMQTT(newSimulacion);
})
.catch((error) => {
    console.error('Error al generar el JSON:', error);
});
```

*Figura 43. Llamada a la función sendMessageMqtt() al generar una nueva simulación. (Fuente propia)*

Una vez definida la estructura de conexión, se procedió a configurar un broker en la plataforma HiveMQ como entorno de prueba para enviar y recibir mensajes. Este paso fue esencial para verificar el correcto funcionamiento de la comunicación MQTT antes de integrarlo completamente en el sistema. Para comenzar, se inició sesión en la plataforma HiveMQ, donde se creó un nuevo cluster que serviría como el punto central para la transmisión y recepción de mensajes.

The screenshot shows the 'Your Clusters' section of the HiveMQ web interface. At the top, there are two buttons: 'Create New Cluster' and 'Quick Start'. Below this, a cluster card is displayed for a cluster named 'Serverless'. The card indicates it is 'FREE' and 'Running'. It provides the URL: b1f3d09eed3e4e998e98502c5567a212.s1.eu.hivemq.cloud and the port (TLS): 8883. The 'Started' timestamp is 2024-11-18T18:09:27Z. A 'Manage Cluster' button is located at the bottom left of the card.

*Figura 44. Cluster creado en la página de HiveMQ para probar el envío de mensajes. (Fuente propia)*

Con el cluster creado, fue necesario volver al archivo del servicio MQTT en el proyecto para actualizar los parámetros de conexión, asegurando que coincidieran con la configuración del nuevo cluster. Entre estos parámetros se incluyeron el host (dirección del broker), el puerto de comunicación, el clientId, y las credenciales necesarias como el username y password. Esta configuración garantizó que la conexión entre el servicio y el broker pudiera establecerse de manera segura y efectiva.

El siguiente paso fue navegar a la herramienta WebSocket Client [17] proporcionada por HiveMQ, la cual permite realizar pruebas interactivas de conexión y mensajería. En esta interfaz, se introdujeron los parámetros configurados del broker y se estableció la conexión. Una vez conectados, se suscribió al tópico /simulaciones, que había sido previamente definido como el canal de prueba para las simulaciones generadas.

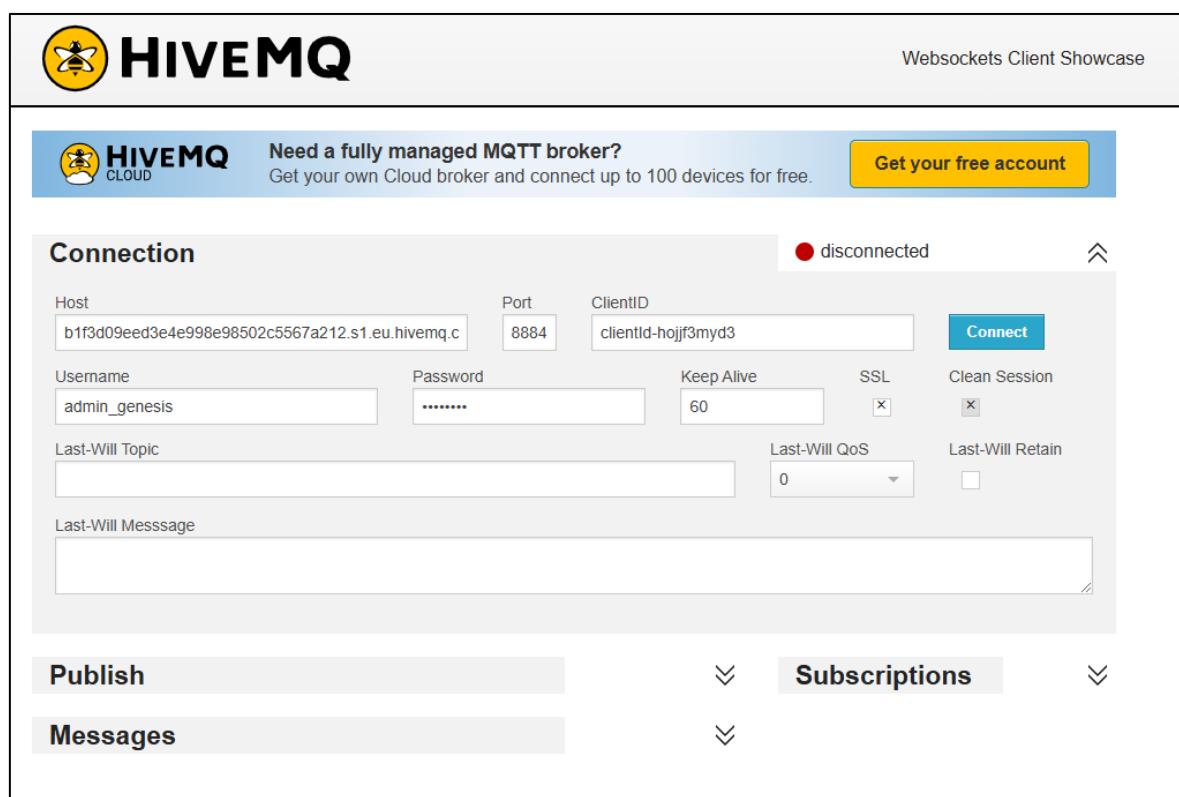


Figura 45. Web Socket Cliente de HiveMQ. (Fuente propia)

Con el sistema configurado, se generó una simulación a modo de prueba, invocando el método sendMessageMQTT para enviar el mensaje al broker. En el cliente WebSocket de HiveMQ, se pudo observar cómo los mensajes eran recibidos en el tópico /simulaciones. Esta visualización confirmó

que las simulaciones generadas estaban siendo correctamente transmitidas desde la aplicación al broker.

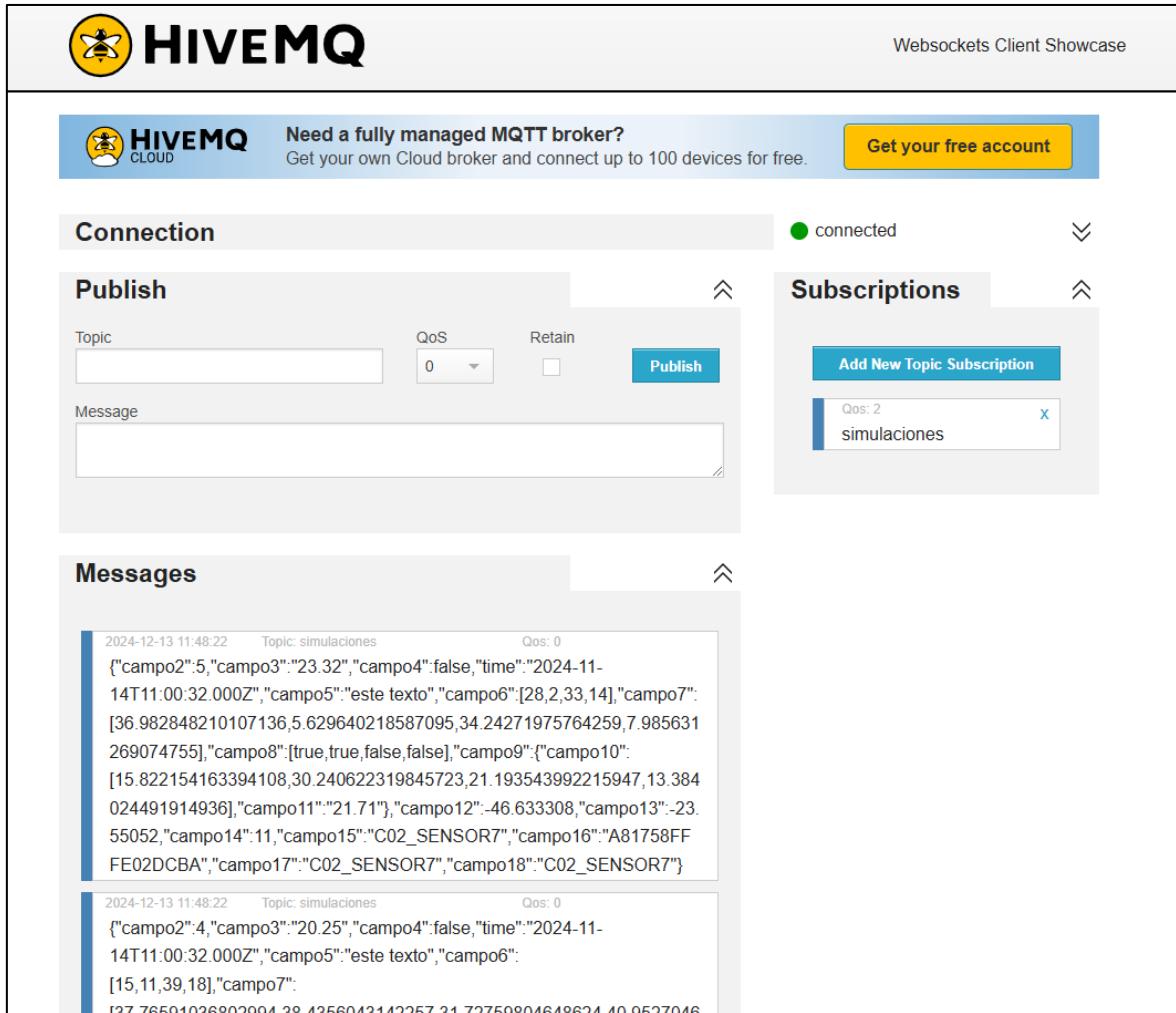


Figura 46. Web Socket Cliente de HiveMQ. (Fuente propia)

Una vez probado su correcto funcionamiento, el siguiente paso será crear un apartado de conexiones, donde podamos definir a que broker y topico queremos enviar el mensaje, ya que actualmente esta definido para siempre enviar al mismo

Una vez probado su correcto funcionamiento, el siguiente paso consiste en desarrollar un apartado de conexiones, donde podamos definir de manera flexible a qué broker y tópico se enviarán los mensajes. Actualmente, la configuración está rígidamente definida para enviar los mensajes siempre al mismo broker y tópico, lo que limita la adaptabilidad del sistema.

El objetivo de esta mejora es proporcionar una mayor flexibilidad, permitiendo que, en cada simulación, se pueda especificar tanto el broker de conexión como el tópico de destino. De esta forma, se optimizará la capacidad del sistema para interactuar con diferentes brokers y tópicos según las necesidades específicas de cada momento, sin la necesidad de realizar modificaciones directas en el código.

## 6.9. Iteración 9. Creación de ruta de conexiones y su componente en el frontend

En esta iteración, el objetivo principal fue la implementación de las rutas API necesarias para gestionar todos los métodos CRUD relacionados con las conexiones. Esta funcionalidad es esencial para la aplicación, ya que permitirá gestionar las simulaciones y su preparación para ser enviadas al software de Kunna, donde serán visualizadas posteriormente.

La pantalla de conexiones permitirá a los usuarios crear nuevas conexiones especificando los parámetros del broker (como URL, puerto, etc.) y el tópico al que se enviarán los mensajes. Una vez gestionadas las conexiones, el usuario podrá seleccionar la conexión deseada al ejecutar una simulación. De esta forma, los mensajes generados se enviarán al broker y tópico seleccionados.

Al igual que en fases anteriores, se comenzó con la fase de análisis para obtener los requerimientos y posteriormente pasaremos a su implementación.

### 6.9.1. Fase de análisis

Anteriormente, el sistema no contaba con una sección dedicada a la gestión de conexiones, lo que limitaba la flexibilidad para configurar de manera dinámica los brokers y tópicos a los que se enviaban los mensajes. En esta fase, se propone implementar una interfaz que permita a los usuarios crear, editar y eliminar conexiones, con el fin de que puedan configurar distintos brokers y tópicos de forma sencilla.

El propósito principal de esta sección es ofrecer una solución flexible que permita al usuario seleccionar la conexión adecuada para cada simulación. Así, se evitara la necesidad de modificar el código o los parámetros del sistema cada vez que se desee cambiar el broker o el tópico de destino.

Para ello, se implementará un conjunto de operaciones CRUD que permitirán gestionar las conexiones de manera eficiente. Estas operaciones permitirán crear nuevas conexiones, editar las existentes, eliminar las que ya no sean necesarias y obtener un listado completo de todas las conexiones configuradas. Además, se incorporará un mecanismo que asociará cada conexión a un usuario específico, asegurando que cada usuario tenga acceso solo a sus propias configuraciones, mientras que los administradores podrán acceder a todas las conexiones disponibles.

La interfaz de usuario será similar a las ya implementadas en el sistema, con una página inicial que muestre un listado de las conexiones configuradas. Este listado incluirá botones para editar, eliminar y crear nuevas conexiones, y contará con una funcionalidad de búsqueda que facilitará la localización de conexiones específicas.

La página de creación de una nueva conexión permitirá seleccionar primero el tipo de conexión que se desea configurar: MQTT o API. Dependiendo de la opción elegida, el formulario mostrará campos específicos. Si se selecciona MQTT, se podrán ingresar parámetros como el nombre de la conexión, URL del broker, puerto, usuario, contraseña y tópico. Si se elige API, se habilitarán campos adicionales como la URL de la API y token de autenticación. No obstante, debido a limitaciones de tiempo, en esta fase solo se implementará la opción de MQTT, dejando la posibilidad de configurar API planteada para más adelante.

#### 6.9.2. Fase de implementación

Una vez definidos los requerimientos, se procedió con la implementación de la gestión de conexiones, comenzando por el backend. En primer lugar, se creó un modelo de conexión que define los campos necesarios para almacenar la información de cada conexión. Este modelo incluye un campo de tipo string para el nombre de la conexión, un campo para la URL del broker, otro para el puerto, así como campos adicionales para el usuario, contraseña y tópico. Además, se incluyó una clave foránea denominada userId, que establece una relación con el modelo User para asociar cada conexión al usuario que la creó. Esta relación se establecerá utilizando Connections.belongsTo(User, { foreignKey: 'userId' }).

En cuanto al controlador, se desarrollarán los siguientes métodos:

- newConnection(): Este método se encargará de crear una nueva conexión. Recibirá en el cuerpo de la solicitud el nombre de la conexión, el tipo de conexión y los parámetros de la conexión, además de recuperar el ID del usuario desde el token de autenticación.
- deleteConnection(): Este método recibirá un ID como parámetro y, si la conexión existe, procederá a eliminarla de la base de datos.

- `updateConnection()`: Este método recibirá un ID como parámetro y, si la conexión existe, procederá a actualizar los datos de la conexión.
- `getAllConnections()`: Este método obtendrá el rol y el ID del usuario. Si el usuario tiene un rol de administrador, se devolverán todas las conexiones. En caso contrario, si el usuario tiene un rol básico, se filtrarán las conexiones por su ID, devolviendo únicamente aquellas que ha creado.
- `getConnectionById()`: Este método recibirá un ID como parámetro y devolverá la conexión correspondiente a ese ID.

Una vez implementados los métodos en el backend, se procederá a la creación del componente y sus interfaces en el frontend, denominado Conexiones. La página inicial mostrará un listado de todas las conexiones configuradas, presentadas en una tabla. Cada fila incluirá información clave sobre la conexión, como el nombre, el tipo y las opciones configuradas. En la última columna de la tabla habrá un único botón de Eliminar, el cual, al ser pulsado, llamará al método `deleteConnection` del servicio para eliminar la conexión seleccionada. Además, al hacer clic en cualquier conexión de la tabla, se abrirá el panel de edición, que redirigirá al componente correspondiente para modificar los detalles de la conexión. Para facilitar la búsqueda de conexiones específicas, se incorporará un buscador, y también se incluirá un botón de Crear que llevará al usuario a una página donde podrá ingresar los parámetros necesarios para configurar una nueva conexión.

The screenshot shows a web application interface for managing connections. At the top, there is a navigation bar with icons for Simulaciones, Sensores, Conexiones (highlighted in blue), and Gestión usuarios. On the far right, there is a user profile icon and the text 'bge3'. Below the navigation bar, the title 'Lista conexiones' is displayed, followed by a search bar and a 'Nuevo' button. The main content area is a table listing four connections:

Usuario	ID	Nombre	Tipo	Opciones
adm2	4	APIConnection	API	({"URL": "https://ingest.kunna.es", "username": "", "password": "...", "headers": {}, "body": ""})
adm2	8	Hive MQ - EPS II	MQTT	({"URL": "wss://bf13d09eed3e4e998e98502c5567a212.st.eu.hivemq.com:8084", "username": "bge3", "password": "...", "headers": {"Content-Type": "application/json"}, "body": ""})
bge3	10	SensorMQTT1	MQTT	({"URL": "wss://broker.example.com:8883/mqtt/8084", "username": "bge3", "password": "...", "headers": {"Content-Type": "application/json"}, "body": ""})
adm2	11	Hive MQ - Aulario II	MQTT	({"URL": "wss://bf13d09eed3e4e998e98502c5567a212.st.eu.hivemq.com:8084", "username": "bge3", "password": "...", "headers": {"Content-Type": "application/json"}, "body": ""})

At the bottom left of the page, there is a small footer note: 'Genesis © 2024'.

*Figura 47. Página de gestión de conexiones de la aplicación. (Fuente propia)*

Los componentes de edición y creación de conexiones compartirán el mismo formulario. En el caso de la creación, el formulario aparecerá vacío y llamará a la función `newConnection` del servicio para registrar una nueva conexión. Por otro lado, en la edición, el formulario se completará con los datos

correspondientes de la conexión seleccionada y llamará a la función updateConnection, pasándole el ID de la conexión. Como se muestra en la interfaz, el formulario incluirá campos para ingresar el nombre y el tipo de conexión, ya sea MQTT o API. Dependiendo de la opción elegida, el formulario mostrará los campos correspondientes. Si se selecciona MQTT, se solicitarán parámetros como la URL del broker, el cliente, el usuario, la contraseña y el tópico. Si se selecciona API, se mostrará la URL y el header.

Nombre:

Tipo de conexión: MQTT

URL (Puerto incluido): wss://broker.example.com:port/mqtt

ClientID: mqtt\_d9lpkq3xs5c

Username:

Password:

Tópico:

Guardar

Figura 48. Página de crear conexión de la aplicación. (Fuente propia)

Una vez creados los componentes y comprobado su funcionamiento, se gestionó el acceso a la sección de conexiones según el rol de cada usuario. Al igual que con las localizaciones, los usuarios básicos solo podrán ver y manipular las conexiones que ellos mismos hayan creado, mientras que los administradores tendrán acceso a todas las conexiones configuradas. Para ello, se utilizó el mismo enfoque que en la gestión de localizaciones: el método getAllConnections está diseñado para filtrar las conexiones en función del rol del usuario. Si el usuario tiene el rol de administrador, se devolverán todas las conexiones; si el usuario tiene un rol básico, solo se devolverán aquellas conexiones asociadas a su ID de usuario.

Además, se implementó un AuthGuard en las rutas correspondientes para restringir el acceso a usuarios no autenticados. Esto garantiza que solo los usuarios autenticados puedan acceder a la página de conexiones. El AuthGuard se aplica utilizando la propiedad canActivate: [AuthGuard] en las rutas, lo que impide que usuarios no autenticados accedan a estas funcionalidades.

Con el gestor de conexiones ya en funcionamiento, el siguiente paso fue actualizar el componente de simulaciones para integrar la gestión de conexiones. Se añadió una nueva opción en el menú de

simulaciones que permite al usuario elegir a qué conexión enviar las simulaciones. Esta conexión seleccionada se guarda dentro de las opciones de la simulación, y cuando se inicia una simulación, el servicio correspondiente obtiene esta conexión y utiliza los datos configurados para enviar el mensaje a través del servicio de MQTT.

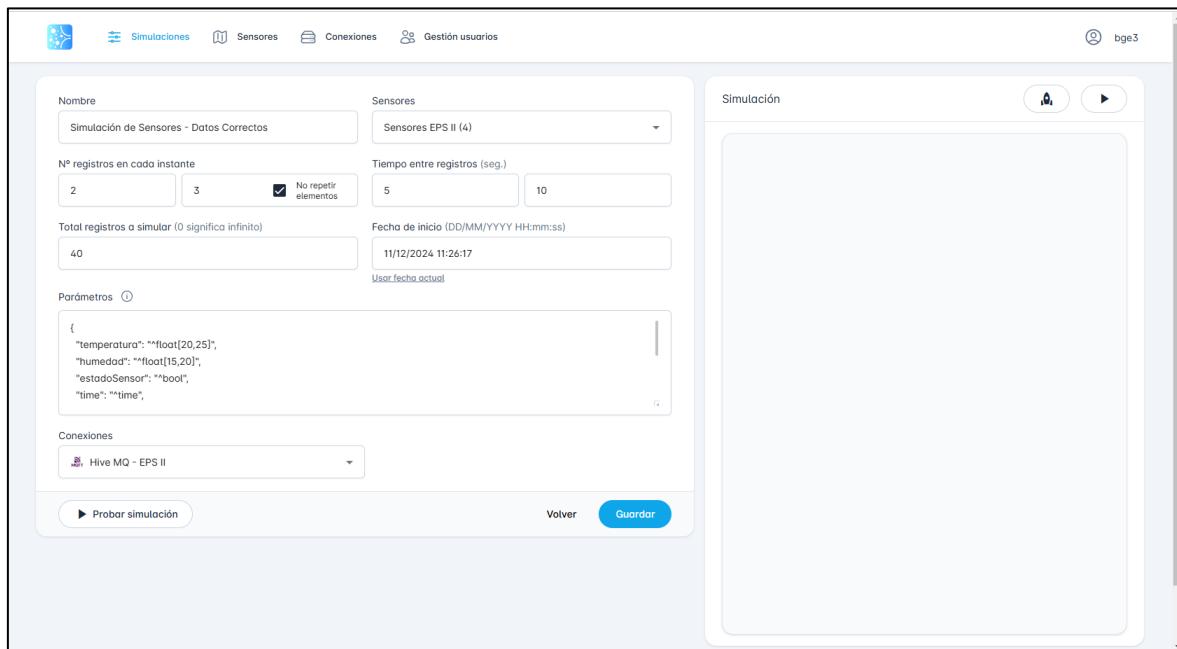


Figura 49. Panel de simulacion con la nueva opción para seleccionar la conexión. (Fuente propia)

Para implementar esta funcionalidad, fue necesario modificar los métodos involucrados en el envío de mensajes. En concreto, el método `sendMessage()` se actualizó para que reciba los datos de la conexión de la simulación y los pase al servicio de MQTT al momento de enviar el mensaje.

```

// Enviar mensaje con las simulaciones generadas por MQTT o API
sendMessage(message: string, connectionId: number): void {
    const messageMQTT = JSON.stringify(message);

    this._conexionesService.getConnectionById(connectionId).subscribe(
        (conexion) => {
            // Conexión y envío por MQTT
            if (conexion.type == 0) {
                this.mqttService.sendMessageMqtt(
                    conexion.options.URL,           // Broker URL
                    conexion.options.clientId,      // Client ID
                    conexion.options.username,      // Username
                    conexion.options.password,      // Password
                    conexion.options.topic,         // Tópico
                    messageMQTT                   // Mensaje
                );
            }
            // Conexión y envío por API
            } else if (conexion.type == 1) {
                this.apiService.sendMessageApi(
                    conexion.options.URL,
                    conexion.options.header,
                    messageMQTT
                );
            }
        }, (error) => {
            console.error('Error al obtener la conexión:', error); // Manejo de error si no se obtiene la conexión
        }
    );
}

```

Figura 50. Método `sendMessage()` actualizado, pasando los datos de la conexión de la simulación al enviar el mensaje. (Fuente propia)

Por el lado de `mqtt.service`, hemos actualizado el servicio creando un nuevo método llamado `sendMessageMqtt`, que ahora recibe los parámetros necesarios para establecer una conexión con el broker MQTT y enviar un mensaje a un tópico específico. Anteriormente, el broker y el tópico se definían de manera fija, pero con esta actualización, la conexión y el envío de mensajes se

adaptan según los parámetros proporcionados. Este método toma los siguientes valores: url, clientId, username, password, topic y message.

```

sendMessageMqtt(url: string, clientId: string, username: string, password: string, topic: string, message: string): void {
    // Crear la conexión MQTT
    const mqttClient = mqtt.connect(url, {
        clientId: clientId,
        username: username,
        password: password,
        clean: true,           // Sesión limpia
        reconnectPeriod: 1000, // Intentar reconectar cada 1 segundo
        keepalive: 60          // Mantener conexión activa con un ping cada 60 segundos
    });

    // Manejar eventos del cliente
    mqttClient.on('connect', () => {
        console.log('Conexión establecida con el broker MQTT');

        // Publicar el mensaje en el tópico indicado
        mqttClient.publish(topic, message, (err) => {
            if (err) {
                console.error('Error al enviar mensaje:', err);
            } else {
                console.log(`Mensaje enviado al tópico ${topic}:`, message);
            }
        });

        // Cerrar la conexión tras enviar el mensaje
        mqttClient.end();
    });
}

// Manejar errores de conexión
mqttClient.on('error', (err) => {
    console.error('Error en la conexión MQTT:', err);
    mqttClient.end(); // Cierra la conexión si ocurre un error
});

// Opción para manejar desconexiones
mqttClient.on('offline', () => {
    console.warn('Cliente MQTT desconectado');
});
}

```

*Figura 51. Método sendMessageMqtt que recibe los parámetros de la conexión, establece la conexión y envía el mensaje. (Fuente propia)*

De esta manera, la conexión y el envío de mensajes a partir de ahora se realiza al broker y tópico seleccionados en la simulación.

Para probar su funcionamiento, hemos creado dos conexiones utilizando el broker de prueba HiveMQ, cada una con un tópico distinto: eps2 y aulario2. Luego, hemos iniciamos dos simulaciones diferentes, cada una asociada a una de estas conexiones. Como se puede observar,

es posible enviar simulaciones de forma simultánea al mismo broker, pero asignando diferentes tópicos.

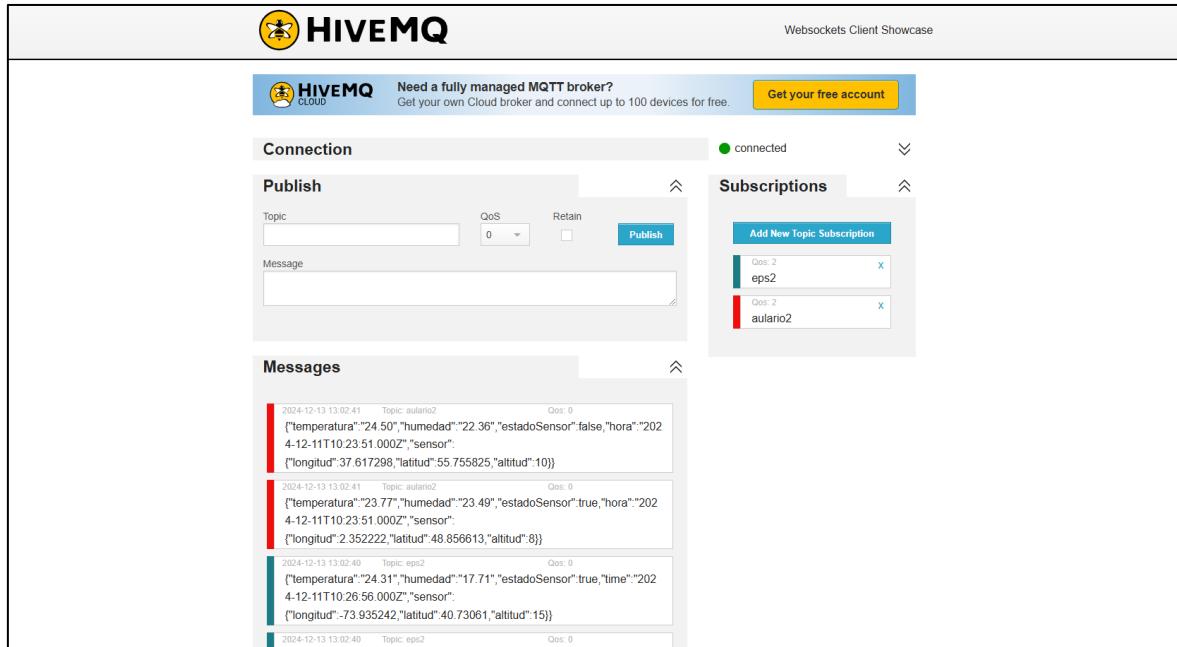


Figura 52. Prueba de envío de mensajes al mismo broker pero distinto tópico. (Fuente propia)

Este enfoque ofrece una gran flexibilidad, ya que permite arrancar tantas simulaciones como se desee, enviándolas al broker y tópico que queramos. De este modo, es posible gestionar y enviar simulaciones a múltiples brokers y tópicos de manera paralela, adaptándose a diversas necesidades.

## 6.10. Iteración 10. Mejoras y refinamiento de la aplicación

En esta iteración, el objetivo principal fue pulir la aplicación y añadir pequeñas implementaciones para lograr un producto completo y funcional. Entre las áreas a mejorar se incluyeron la implementación de la paginación, la resolución de problemas de sincronización entre las simulaciones iniciadas en el menú y su estado individual, la unificación de los botones y el estilo de la interfaz, así como la corrección de errores que surgían al intentar probar una simulación mientras estaba en ejecución.

Al igual que en fases anteriores, se comenzó con un análisis detallado para obtener los requerimientos, y a continuación, se procedió a su implementación..

### 6.10.1. Fase de análisis

En esta fase, el objetivo principal fue identificar los requerimientos necesarios para mejorar la aplicación y solucionar los problemas existentes. El proceso comenzó con una prueba interna de funcionamiento, seguida de una reunión con el tutor, quien arrancó la aplicación en su propio equipo para probarla y proporcionar retroalimentación. Durante esta revisión, se identificaron varios aspectos a mejorar y surgieron algunos errores menores durante la ejecución.

Uno de los problemas mencionados fue la falta de paginación en el listado de simulaciones, lo que en un futuro podría afectar a la experiencia del usuario cuando el número de simulaciones sea elevado. Otro aspecto relevante que se analizó fue la interfaz de usuario. Durante esta revisión, se detectaron inconsistencias en el diseño de los botones y otros elementos visuales, por lo que se propuso unificar el estilo de todos los botones y componentes para mejorar la experiencia del usuario y la coherencia de la interfaz.

Además, se identificaron errores de sincronización entre las simulaciones arrancadas desde el menú y su estado individual, lo que ocasionaba incoherencias en la visualización de la información.

Asimismo, se detectaron errores específicos que ocurrían cuando se intentaba probar una simulación mientras esta ya se encontraba en ejecución. Estos errores necesitaban ser corregidos para garantizar que las simulaciones pudieran ejecutarse y probarse de manera confiable y sin fallos.

### 6.10.2. Fase de implementación

Una vez definidos los requerimientos, se procedió a la implementación comenzando con la paginación del listado de simulaciones. Se añadió un menú en la parte inferior donde el usuario puede seleccionar cuántos elementos desea visualizar a la vez (5, 10 o 25), además de incorporar flechas para navegar entre las páginas. Esto permite una visualización más organizada y eficiente de las simulaciones, especialmente cuando hay un gran número de ellas.

A continuación, se unificó la interfaz. Antes, cada elemento en el listado contaba con un botón de eliminar y otro de editar. Ahora, se eliminó el botón de edición, y al hacer clic en un elemento, se abrirá directamente su vista de edición. También se realizaron ajustes en los márgenes y el

espaciado, con el fin de optimizar el espacio en pantalla y de esta manera, tener la máxima información a la vista.

Por otro lado, se detectaron errores al ejecutar una simulación desde el listado y luego abrirla individualmente, lo cual fue corregido. Como resultado, la interacción se volvió mucho más fluida y potente: ahora se pueden arrancar tantas simulaciones como se desee desde el listado, y al hacer clic en una de ellas, se abre su vista individual, permitiendo observar su ejecución y controlarla de manera independiente. En la siguiente captura se puede observar parte de estas mejoras, como la barra de paginación, junto con varias simulaciones en ejecución al mismo tiempo, mostrando su progreso, y con la opción de pausarlas o detenerlas.

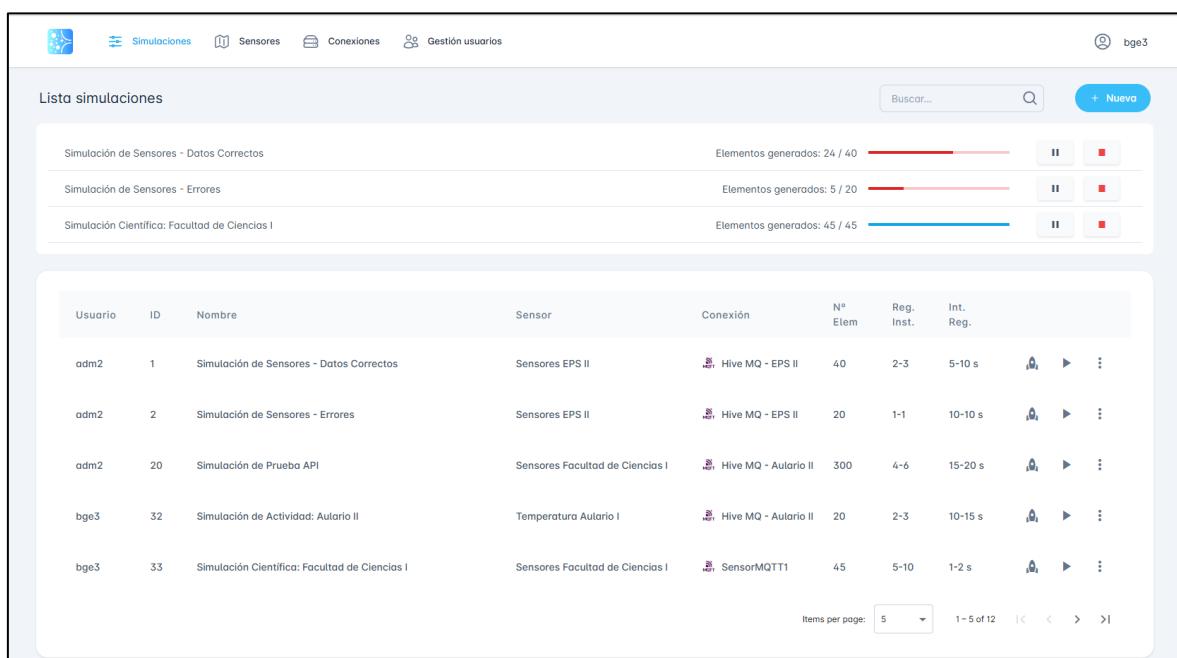


Figura 53. Página de simulaciones en ejecución. (Fuente propia)

También se corrigió un error que ocurría al intentar probar una simulación ya en ejecución. Dado que el panel derecho muestra tanto las pruebas como la simulación en vivo, surgían problemas de interacción entre ambas funcionalidades. Este inconveniente fue solucionado, mejorando la experiencia del usuario y asegurando que las pruebas y simulaciones en vivo puedan gestionarse de manera independiente sin conflictos.

Finalmente, se dedicó tiempo a realizar pequeños retoques y corregir errores menores, lo que permitió dejar la aplicación completamente unificada y funcional.

## 1. Pruebas y validación

Una vez finalizado el desarrollo, se llevó a cabo un periodo de pruebas y validación de la aplicación con el objetivo de prepararla para la demo. Durante esta fase, se intentó recrear un entorno real, simulando el uso de la aplicación en condiciones similares a las que se enfrentarían los usuarios finales. Para ello, se crearon varios componentes dentro de la aplicación, como diferentes usuarios, colecciones de sensores representando las que se encontrarían en el campus, y múltiples conexiones a distintos brokers y tópicos.

Se generaron diversas simulaciones que enviaban distintos tipos de datos a varios brokers y tópicos, lo que permitió verificar el funcionamiento correcto de la aplicación en diferentes escenarios. Esta fase de pruebas fue crucial para garantizar su correcto funcionamiento antes de la demo final.

De cara a la demo, se crearon tres simulaciones: dos dirigidas al mismo tópico, una con datos correctos (temperaturas entre 20 y 25 grados) y otra con datos erróneos (temperaturas entre 50 y 60 grados), simulando un escenario en el que el broker recibe una mezcla de valores reales e irreales. Este escenario podría ser útil en el futuro para implementar un algoritmo que descarte los valores erróneos. Además, se configuró una simulación adicional que enviaba datos de otro sensor a un tópico distinto, lo que permitió observar cómo los mensajes de diferentes tópicos se alternaban en la cola de mensajes del broker.

Por cuestiones de tiempo, no se ha podido probar la aplicación en el entorno real para el cual está diseñada. Sin embargo, en un futuro próximo, el tutor José Vicente y su equipo integrarán la aplicación en el ecosistema de Kunna, y se probará en un entorno real.

Este paso no solo permitirá validar la aplicación en un entorno real, sino que también garantizará su utilidad a largo plazo. Al integrarse en el ecosistema de Kunna, la aplicación se convertirá en una herramienta funcional y valiosa para la universidad, contribuyendo a mejorar la gestión del campus. De esta manera, el proyecto no se limita a una simple demostración, sino que continuará siendo útil en el entorno académico.

## 2. Resultados

Aunque la aplicación aún no está en producción ni ha sido probada en un entorno real, está planteada para integrarse en el ecosistema de Kunna y ser de utilidad para el tutor Jose Vicente y su equipo. Esto permitirá su uso en el campus, facilitando la generación de simulaciones y mejorando la gestión dentro del proyecto Smart University. Al reducir la dependencia de sensores reales, la aplicación contribuirá a realizar pruebas y optimizar la gestión del campus.

En cuanto a los objetivos del proyecto, se puede afirmar que han sido cumplidos con éxito, ya que se ha resuelto la necesidad del cliente, partiendo de unos requerimientos iniciales y proponiendo soluciones a medida durante el desarrollo para lograr el producto final.

Este proyecto ha sido una gran oportunidad para aplicar los conocimientos y competencias adquiridos durante la carrera a través de diversas asignaturas del grado de Ingeniería Multimedia [25]. Me ha permitido enfrentarme a un entorno real, donde existe una necesidad concreta, y seguir un proceso ingenieril para encontrar la solución adecuada.

### 3. Conclusiones y trabajo futuro

Una vez finalizado el proyecto, puedo afirmar que se han cumplido los objetivos propuestos al inicio. Se ha desarrollado una aplicación capaz de generar simulaciones en tiempo real, como temperatura, humedad y niveles de CO<sub>2</sub>, que pueden enviarse a Kunna para su visualización y análisis. La herramienta ofrece una solución que facilita las pruebas y validaciones sin depender exclusivamente de datos reales.

A nivel personal, estoy muy satisfecho con el resultado. Durante el proceso, se ha logrado solucionar una problemática real y, gracias a reuniones periódicas con el tutor, se fueron definiendo los pasos necesarios para avanzar en el desarrollo. Esto permitió realizar ajustes y mejoras iterativas hasta obtener un producto útil que cumple con las expectativas iniciales. Además, este proyecto ha supuesto un aprendizaje significativo en áreas como el desarrollo de aplicaciones web, la gestión de proyectos y el uso de nuevas tecnologías como MQTT.

Aunque se ha abordado la mayor parte del proyecto, por motivos de tiempo, algunas mejoras y funcionalidades han quedado pendientes. Por ejemplo, en el apartado de conexiones, se ha implementado el envío de mensajes mediante MQTT, quedando planteado, aunque sin implementar, la opción de enviar mensajes a través de una API.

Por último, ante la falta de pruebas en el entorno real en el que se integrará el proyecto, podrían surgir nuevas necesidades o mejoras adicionales. Estas requerirían ser abordadas para refinar y ajustar la aplicación hasta que esté completamente operativa y lista para su uso.

## Referencias

1. MIRO | The Innovation Workspace. Disponible en: <https://miro.com/>
2. GitHub: Let's build from here. GitHub. Disponible en: <https://github.com/>
3. BrunoEscudero - GitHub - BrunoEscudero/Simulacion-ETL-TFG: Este proyecto crea una herramienta de simulación de datos para la plataforma Kunna del proyecto Smart University de la Universidad de Alicante. Genera datos simulados de sensores (temperatura, humedad, CO2) y los envía a Kunna para mejorar la visualización y análisis, optimizando así la gestión del campus., GitHub. Disponible en: <https://github.com/BrunoEscudero/Simulacion-ETL-TFG>.
4. Visual Studio Code - Code editing. Disponible en: <https://code.visualstudio.com/>
5. GitHub Desktop | Simple collaboration from your desktop. Disponible en: <https://github.com/apps/desktop/>
6. XAMPP Installers and Downloads for Apache. Disponible en: [https://www.apachefriends.org/es/index.html/](https://www.apachefriends.org/es/index.html)
7. Node.js — Run JavaScript everywhere. Disponible en: <https://nodejs.org/en/>
8. Fuse Theme Family. FuseTheme. Disponible en: <https://fusetheme.com/>
9. Express - Node.js web application framework. Disponible en: <https://expressjs.com/>
10. Postman API Platform. Disponible en: <https://www.postman.com/>
11. Eclipse Mosquitto. Disponible en: <https://mosquitto.org/>
12. Kunna - Universidad de Alicante. UA Smart University, UA Smart University. Disponible en: <https://smart.ua.es/es/kunna/kunna-la-plataforma.html>
13. Smart University UA - University of Alicante. UA Smart University, UA Smart University. Disponible en: <https://smart.ua.es/>
14. MQTT - The Standard for IoT Messaging. Disponible en: <https://mqtt.org/>
15. ¿Qué es MQTT? Definición y detalles. <https://www.paessler.com/es/it-explained/mqtt>
16. HiveMQ – The Most Trusted MQTT platform to Transform Your Business. Disponible en: <https://www.hivemq.com/>
17. MQTT Websocket Client. Disponible en: <https://www.hivemq.com/demos/websocket-client/>
18. Manage your team's projects from anywhere | Trello. Disponible en: <https://trello.com/>
19. Clockify - Software de control del tiempo GRATIS, Clockify. Disponible en: <https://clockify.me/es/>
20. Angular. Disponible en: <https://angular.dev/>

21. Ingeniería Multimedia - Itinerario Gestión de Contenidos. Universidad de Alicante.  
Disponible en: <https://eps.ua.es/es/ingenieria-multimedia/gestioncontenidos/que-es-abp.html>
22. JSON. Disponible en: <https://www.json.org/json-es.html>
23. Cruz, M. R. "¿Qué son las «smart cities»?", BBVA NOTICIAS, 22 mayo. Disponible en: <https://www.bbva.com/es/sostenibilidad/las-smart-cities/>
24. ¿Qué es la metodología ágil? ¿Para qué sirve? Zendesk. Disponible en: <https://www.zendesk.com.mx/blog/metodologia-agil-que-es/>
25. De Alicante, U. Grado en Ingeniería Multimedia, Grado En Ingeniería Multimedia.  
Disponible en: <https://web.ua.es/es/grados/grado-en-ingenieria-multimedia/>
26. Tailwind CSS - Rapidly build modern websites without ever leaving your HTML. Tailwind CSS. Disponible en: <https://tailwindcss.com/>
27. Internet de las cosas, Wikipedia, la Enciclopedia Libre. Disponible en: [https://es.wikipedia.org/wiki/Internet\\_de\\_las\\_cosas](https://es.wikipedia.org/wiki/Internet_de_las_cosas)

## Apéndice

En este apéndice se incluye el enlace al repositorio del proyecto, así como la estructura de este y las instrucciones necesarias para configurarlo y desplegarlo.

El repositorio puede encontrarse en GitHub en el siguiente enlace:  
<https://github.com/BrunoEscudero/TFG-Simulacion-ETL>.

El repositorio está compuesto principalmente por dos carpetas: backend y frontend. La primera carpeta contiene todo el código relacionado con las funciones de la API de la aplicación y con la conexión de la base de datos. Dentro de esta hay una carpeta llamada bd donde se encuentra el script SQL con la estructura de la base de datos, llamado bdd\_prueba.sql el cual se puede importar en cualquier base de datos relacional.

Por otro lado, la carpeta frontend incluye todos los archivos y el código relacionado con el lado del cliente de la aplicación, como los componentes, los servicios utilizados y la plantilla empleada para la interfaz de usuario.

Para realizar pruebas iniciales, puedes utilizar el siguiente usuario con permisos de administrador:

- Usuario: adm2
- Contraseña: ua

Para arrancar el proyecto, lo primero que necesitas hacer es clonar el repositorio en tu máquina local. Una vez clonado el repositorio, el siguiente paso es configurar el entorno de desarrollo adecuado para que la aplicación funcione correctamente. Para el backend, necesitarás un entorno con Apache y MySQL, por lo que se recomienda utilizar XAMPP, que proporciona ambos servicios.

Cuando XAMPP esté instalado, abre su panel de control y asegúrate de iniciar los servicios de Apache y MySQL. Con estos servicios activos, abre tu navegador y accede a phpMyAdmin a través de la URL <http://localhost/phpmyadmin/>. Desde allí, crea una nueva base de datos llamada tfg. Luego, ve a la pestaña "Importar" y selecciona el archivo bdd\_prueba.sql que se encuentra en la carpeta bd del proyecto. Este archivo contiene la estructura necesaria para una base de datos a modo de prueba de la aplicación.

Una vez que el entorno del backend esté listo, pasa a configurar el frontend. Abre una terminal, navega hasta la carpeta frontend del proyecto y ejecuta el comando npm install para instalar todas las dependencias necesarias para el frontend.

A continuación, abre dos terminales: una en la carpeta del frontend y otra en la carpeta node del backend. En ambas terminales, ejecuta el siguiente comando para iniciar la aplicación: `npm start`. Esto arrancará tanto el frontend como el backend de la aplicación.

Finalmente, al ingresar al enlace `http://localhost:4200/`, podrás ver cómo la aplicación se ha iniciado correctamente. Desde allí, podrás probar y utilizar todas sus funcionalidades.