Componente de apoyo para el análisis y visualización de los datos de dispositivos IoT en la arquitectura Smart Campus UIS

Andrés Felipe Rojas Santos

Trabajo de Grado para optar al título de Ingeniero de Sistemas

Director

Gabriel Rodrigo Pedraza Ferreira

Doctor en Ciencias de la Computación

Codirector

Henry Andrés Jiménez Herrera

Maestría en Ingeniería de Sistemas

Universidad Industrial de Santander

Facultad de Ingenierías Fisicomecánicas

Escuela de Ingeniería de Sistemas e Informática

Bucaramanga

2023

Dedicatoria

A mí mismo, por demostrarme de lo que soy capaz al desarrollar este proyecto en un tiempo corto y con muchas ganas de aprender más cada día.

Agradecimientos

A mi familia por apoyarme y alentarme a seguir, sabiendo que yo era capaz de vencer a la universidad.

A los directores de este proyecto por su orientación en el desarrollo de este proyecto de grado.

Tabla de Contenido

Lista de Figuras

Resumen

Título: Componente de apoyo para el análisis y visualización de los datos de dispositivos IoT en la arquitectura Smart Campus UIS [*]

Autores: Andrés Felipe Rojas Santos [**]

Palabras Clave: Aplicación web, Dashboard, gráficas, datos.

Descripción: En la actualidad es recurrente el término internet de las cosas (IoT) o campus inteligente (Smart Campus), dos términos que juntos nos brindan ideas para soluciones tecnológicas de alto impacto a la comunidad universitaria, conectar sistemas como la iluminación y las cerraduras de las puertas, permiten crear una experiencia fluida de interconexión para los estudiantes, la facultad y el personal. Mediante el uso de datos y análisis, los campus inteligentes pueden optimizar procesos y recursos de sus instalaciones. Esto hizo que fuera diseñado una arquitectura de campus universitario denominada, Smart Campus UIS, un proyecto realizado para diseñar e implementar un campus inteligente en la Universidad Industrial de Santander. Actualmente, el sistema permite la administración de las aplicaciones, dispositivos, procesos y demás que hacen parte de este sistema. Para este proyecto se plantea el diseño de una aplicación Web, con un panel de control para obtener y visualizar un monitoreo en tiempo real o histórico de los datos suministrados por los distintos sensores, como lo son los sensores de temperatura, gas, consumo energético, entre otros. Como consecuencia del trabajo realizado, se considera óptimo hacer pruebas para la integración entre la arquitectura y esta aplicación, diseñando casos de uso para diferentes sensores que podrían ser comunes y del interés de la comunidad universitaria.

---

[*] Trabajo de grado

[**] Facultad de Ingenierías Fisicomecánicas. Escuela de Ingeniería de Sistemas e Informática. Director: Gabriel Rodrigo Pedraza Ferreira

Abstract

Title: Support Component for IoT Device Data Analysis and Visualization in the Smart Campus UIS Architecture [*]

Authors: Andrés Felipe Rojas Santos [**]

Keywords: Web application, Dashboard, charts, data.

Description: Nowadays, the term “Internet of Things” (IoT) and “Smart Campus” are commonly used phrases that provide ideas for high-impact technological solutions for the university community. Connecting systems such as lighting and door locks enables a seamless interconnected experience for students, faculty, and staff. Through the use of data and analysis, smart campuses can optimize processes and resources within their facilities. This led to the design of a university campus architecture called Smart Campus UIS, a project aimed at designing and implementing a smart campus at the Industrial University of Santander. Currently, the system allows for the management of applications, devices, processes, and other components of this system. For this project, the design of a web application with a control panel is proposed to obtain and visualize real-time or historical monitoring of data provided by various sensors, such as temperature sensors, gas sensors, energy consumption, among others. As a result of the work carried out, it is considered optimal to perform integration tests between the architecture and this application, designing use cases for different sensors that may be common and of interest to the university community.

---

[*] Bachelor Thesis

[**] Faculty of Physicomechanical Engineering. School of Systems and Computer Engineering. Director: Gabriel Rodrigo Pedraza Ferreira

Introducción

La evolución constante de la tecnología junto con la aparición de nuevas tendencias como el internet de las cosas (IoT), han generado cambios significativos tanto en los hogares como en los lugares de trabajo o de estudio. La creación de un campus inteligente, en el que diferentes dispositivos recopilan información para su análisis y procesamiento, se ha convertido en una herramienta fundamental para el desarrollo y la toma de decisiones en las instituciones educativas, en este contexto, dentro de la Universidad Industrial de Santander (UIS) se ha desarrollado una arquitectura de "Smart campus", la cual, permite la conexión de dispositivos para el intercambio de datos a través de internet y a su vez la administración de procesos que se accionan dependiendo de los datos registrados por los sensores. Sin embargo, se ha identificado una necesidad importante de mejora en la plataforma, la cual, se orienta hacia la visualización y análisis que permita proporcionar una cómoda interpretación de los datos suministrados por los dispositivos, generando así información más clara y asequible a fin de tomar decisiones de manera más agradable y optimizada.

Como resultado, el propósito de este proyecto de grado se enfocará en la implementación de un componente gráfico adicional a la arquitectura de "Smart campus IoT UIS", que permita una renovación del diseño en la visualización y análisis estadístico de los datos recopilados por los dispositivos, orientado en si hacia la experiencia del usuario. La relevancia de este componente radica en la posibilidad de proporcionar una herramienta de apoyo como complemento para la toma de

decisiones, a través de la interpretación de la información generada. Siguiendo este orden de ideas, se evidencia el impacto del proyecto para mejorar la interacción del usuario con la percepción gráfica en la toma de decisiones, permitiendo la generación de métricas y pronósticos útiles para el lector.

1. Objetivos

1.1. Objetivo general

Desarrollar un componente de software adaptable para la visualización de datos que extienda la plataforma Smart Campus UIS para facilitar el análisis de los datos por parte de los usuarios de la plataforma.

1.2. Objetivos específicos

- Identificar las necesidades en términos de visualización de datos en la plataforma Smart Campus UIS.
- Diseñar el componente de software de visualización de datos que sea adaptable, permitiendo al usuario generar tableros y gráficos de acuerdo con sus necesidades específicas.
- Implementar el componente software de visualización de datos de acuerdo con el diseño realizado.
- Validar el componente desarrollado a través de un conjunto de pruebas funcionales que incluya escenarios de visualización en tiempo real y/o de datos almacenados.

2. Marco Teorico

En este capítulo hablemos de los siguientes términos importantes, que serán utilizados en todo el proyecto y que son importantes para su campo de aplicación.

2.1. Internet of things (IoT)

La Ïnternet de las cosas"(IoT, por sus siglas en inglés) es un concepto que se refiere a la interconexión de dispositivos a través de Internet. La idea es que estos dispositivos sean capaces de comunicarse en su entorno local y global, permitiendo la recolección y el análisis de datos en tiempo real. De esta manera, se dispone del potencial para automatizar procesos y mejorar la eficiencia de las operaciones. (Gracia, 2023)

2.2. Smart campus

La idea detrás de un Smart Campus es utilizar la tecnología para mejorar la eficiencia, seguridad, sostenibilidad y calidad de vida en el campus universitario. Un Smart Campus utiliza la infraestructura de Internet de las cosas (IoT) para conectar y comunicar una variedad de dispositivos y sensores en el campus, permitiendo la recopilación y análisis de grandes cantidades de datos en tiempo real. La información obtenida de estos dispositivos y sensores se puede utilizar para optimizar la comunicación, el uso de recursos, mejorar la seguridad y tomar decisiones informadas en tiempo real.

Un Smart Campus contiene una amplia gama de aplicaciones y tecnologías, como sistemas de iluminación inteligente, sensores de calidad del aire, sistemas de riego y monitoreo del agua, cámaras de seguridad y aplicaciones móviles para la gestión del campus. Estas tecnologías se integran en una plataforma común que permite la monitorización, gestión y análisis de los datos recopilados. Los sistemas de inteligencia artificial y análisis de datos también se pueden utilizar para sacar mayor provecho a la información.(de Málaga, 2020)

2.3. WebSockets

Los WebSockets permiten la comunicación bidireccional entre aplicaciones web y procesos del lado del servidor, generando un túnel de comunicación en ambos sentidos, exponiendo un puerto en el navegador que permite el envío y recepción de información entre el Cliente y el Servidor.

2.4. Bróker de Mensajería y MQTT

Un Bróker de mensajería es un software intermedio que permite la comunicación entre diferentes aplicaciones, facilitando la traducción de datos, puesto que actúa como un agente de transferencia de mensajes, intercambiándolos entre diferentes aplicaciones, pudiendo ser estas aplicaciones: emisores o receptores. También proporciona la validación, transformación y enrutamiento de los mensajes. MQTT es un servicio de mensajería con patrón publicador/suscriptor (pub-sub) que funciona sobre TCP/IP. Es decir, un servidor central, que en MQTT se denomina Bróker, recibe los mensajes, los filtra y los distribuye a los clientes, esto lo hace aplicando un filtrado a los mensajes

que son recibidos desde los publicadores, para discriminar a qué clientes suscritos es entregado, a este filtro se denomina tópico (Topic en inglés).

2.5. Dashboards

Los dashboards para dispositivos IoT son una herramienta fundamental en la administración y gestión de dispositivos conectados a Internet. Son una interfaz gráfica de usuario que les permite monitorear, visualizar y analizar datos en tiempo real. En el contexto de IoT, una aplicación de estas permite a los usuarios controlar y monitorear dispositivos conectados, y analizar los datos recopilados para tomar decisiones informadas. Además, pueden mostrar estadísticas y tendencias históricas, lo que permite a los usuarios detectar patrones y hacer predicciones.

2.6. Entorno distribuido

Un entorno distribuido hace referencia a un sistema en el cual varias computadoras o dispositivos interconectados colaboran entre sí para realizar una tarea o cumplir un objetivo común. En lugar de disponer de una única máquina ejecutando todo el proceso, las tareas se distribuyen entre múltiples dispositivos que trabajan de manera conjunta. Cada uno de estos dispositivos puede tener su propio sistema operativo, recursos y capacidades individuales. Estas máquinas se comunican entre sí a través de protocolos de red y comparten información y recursos para realizar tareas específicas. (Zettler, 2023)

2.7. Gateways

Una puerta de enlace que actúa como un puente, para proporcionar una transferencia de datos de manera pronta y segura entre la información de los dispositivos conectados. Al usar una puerta de enlace, las organizaciones pueden mantener bases de datos y otros orígenes de datos en sus redes locales, pero usar de forma segura en servicios en la nube. (AWS, 2023a)

2.8. API

Las API significa “interfaz de programación de aplicaciones” y son mecanismos que permiten a dos componentes de software comunicarse entre sí mediante un conjunto de definiciones y protocolos. Por ejemplo, el sistema de software del instituto de meteorología contiene datos meteorológicos diarios. La aplicación meteorológica de su teléfono “habla” con este sistema a través de las API y le muestra las actualizaciones meteorológicas diarias en su teléfono.(AWS, 2023a)

2.9. Microservicios

Son un paradigma arquitectónico de desarrollo de software que se caracteriza por la descomposición de una aplicación monolítica en una colección de servicios independientes y autónomos. Cada microservicio representa una unidad funcional específica de la aplicación y opera de manera aislada en su propio entorno de ejecución. Estos microservicios están diseñados para cumplir funciones particulares del negocio y se comunican a través de interfaces definidas, como API HTTP o protocolos de mensajería. La arquitectura de microservicios promueve la independencia,

escalabilidad, flexibilidad y mantenibilidad de los componentes de software, lo que permite un desarrollo y gestión eficiente de recursos.(AWS, 2023b)

2.10. Spring Security

Spring Security es un marco que se enfoca en proporcionar tanto autenticación como autorización a aplicaciones Java. Como en todos los proyectos de Spring, el verdadero poder de Spring Security se encuentra en lo fácil que es extenderlo para cumplir con requisitos personalizados.(Tanzu, 2023)

3. Marco de referencia

El proyecto de Smart Campus IoT UIS es un proyecto que se ha venido desarrollando en proyectos de grados previos a este y que actualmente cuenta con las bases ya desarrolladas de la infraestructura de alta disponibilidad, un Framework para los Gateways, un backend para autenticación y administración de datos, y un frontend de administración. A continuación, describiremos cada uno de estos componentes y su aplicación en el contexto de este proyecto.

3.1. Infraestructura cloud

La base de toda la arquitectura de este proyecto fue planteada en el proyecto de grado titulado *"Definición de una infraestructura cloud de alta disponibilidad en un entorno distribuido para el despliegue de una plataforma IoT"* (Rojas, 2019). El objetivo de este proyecto fue crear un entorno distribuido que pueda soportar multitud de dispositivos conectados, permitiendo la recolección y transmisión masiva de información dentro y fuera de la infraestructura.

3.2. Framework para Gateways

El proyecto *"Diseño de un Framework software extensible para dispositivos tipo Gateway integrados en plataformas IoT para Smart Campus"* (Gutiérrez, 2019) desarrolló el software necesario para los Gateway de la arquitectura; este software, actúa como un servidor de administración para los dispositivos y procesos vinculados con el gateway, adicional a esto, cuenta con la capa-

cidad de almacenar los datos generados por los dispositivos con el fin de guardar localmente la información, permitiendo así, su funcionamiento sin conexión a internet de manera temporal hasta tener la disponibilidad del servidor nuevamente, lo cual, evita perdida de datos, debido a que en general, solo cuando existe la conexión a internet, se sincroniza los datos con el servidor en la nube, además de este servicio, cuenta con una API que permite enviar al Gateway una aplicación compilada de *Java*, esto con el fin de ser ejecutada como un nuevo proceso que permite tener un comportamiento adaptable a las situaciones que se planteen, ya sea, obtener la información de un sensor y en determinados estados accionar un actuador.

3.3. Backend de la arquitectura

El backend de la arquitectura fue realizado en el proyecto de grado *"Diseño del componente software backend orientado a una plataforma IoT diseñada para Smart Campus"* (Arias & Estupiñan, 2019). Este proyecto consta de tres backend intercomunicados siguiendo la idea de un sistema distribuido.

3.3.1. Backend de administración (admin). Este componente proporciona una API para la gestión de los dispositivos, aplicaciones, procesos y usuarios a manera de CRUD; se encarga además del sistema de la autenticación para los usuarios utilizando Spring Security.

3.3.2. Backend de datos (data). Almacenamiento y suministro de datos. Este componente se encarga de gestionar los mensajes enviados por los dispositivos a través del bróker de la arquitectura y de almacenarlos en MongoDB, también proporciona una API para obtener esta-

dísticas de usuarios y un historial de los datos generados por un dispositivo o un proceso.

3.3.3. Backend de procesos (jobs). Se encarga de monitorizar los procesos que se ejecutan en los gateway, para mantener al usuario al informado del estado o señales de alerta de los mismos; igualmente permite la gestión de los procesos previamente mencionados que puede ejecutar un Gateway.

3.4. Front de administración

Es la Aplicación web para la administración de la arquitectura, utilizando las API del backend de datos y administración, podemos gestionar un CRUD de los usuarios, aplicaciones, Gateways, procesos y dispositivos, como a su vez recibir notificaciones procedentes de los procesos mediante las API del backend de procesos. Como contextualización, cuando hablamos de las "aplicaciones" dentro de la arquitectura, nos referimos a un conjunto para los Gateways que en la práctica no tienen mayor utilidad que clasificarlos en grupos.

En la Figura 1 se presenta el diagrama realizado en el proyecto de grado mencionado en el capítulo 3.3 para la arquitectura de Smart Campus UIS, donde "S" son los sensores y "A" los actuadores.

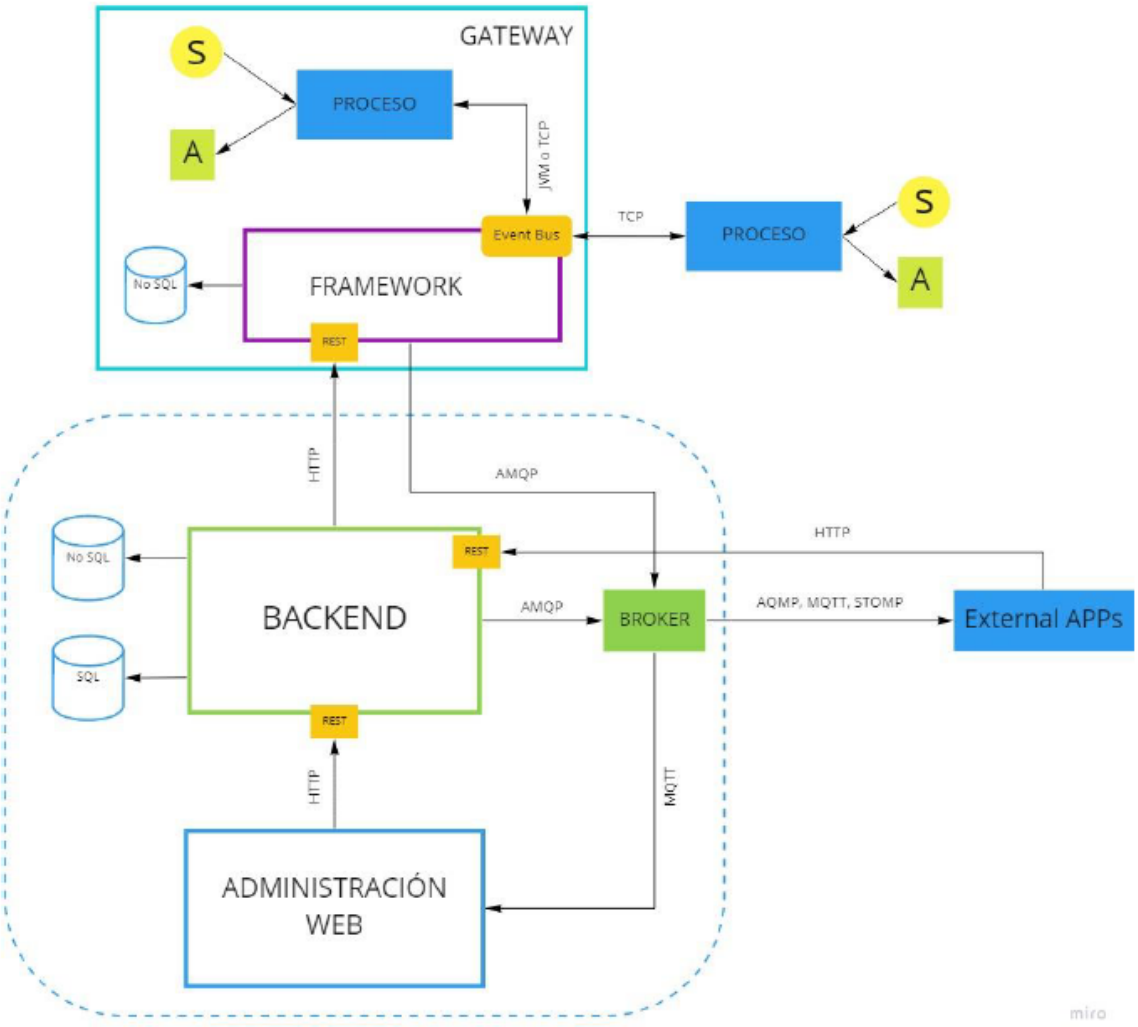

Figura 1. Diagrama de la arquitectura de Smart Campus UIS V1

4. Metodología

En este capítulo se presentará la metodología para el desarrollo de este proyecto, basada en actividades y enfocada en el cumplimiento de los objetivos planteados. La figura 2 ilustra las actividades realizadas y el orden en el que se desarrollaron.

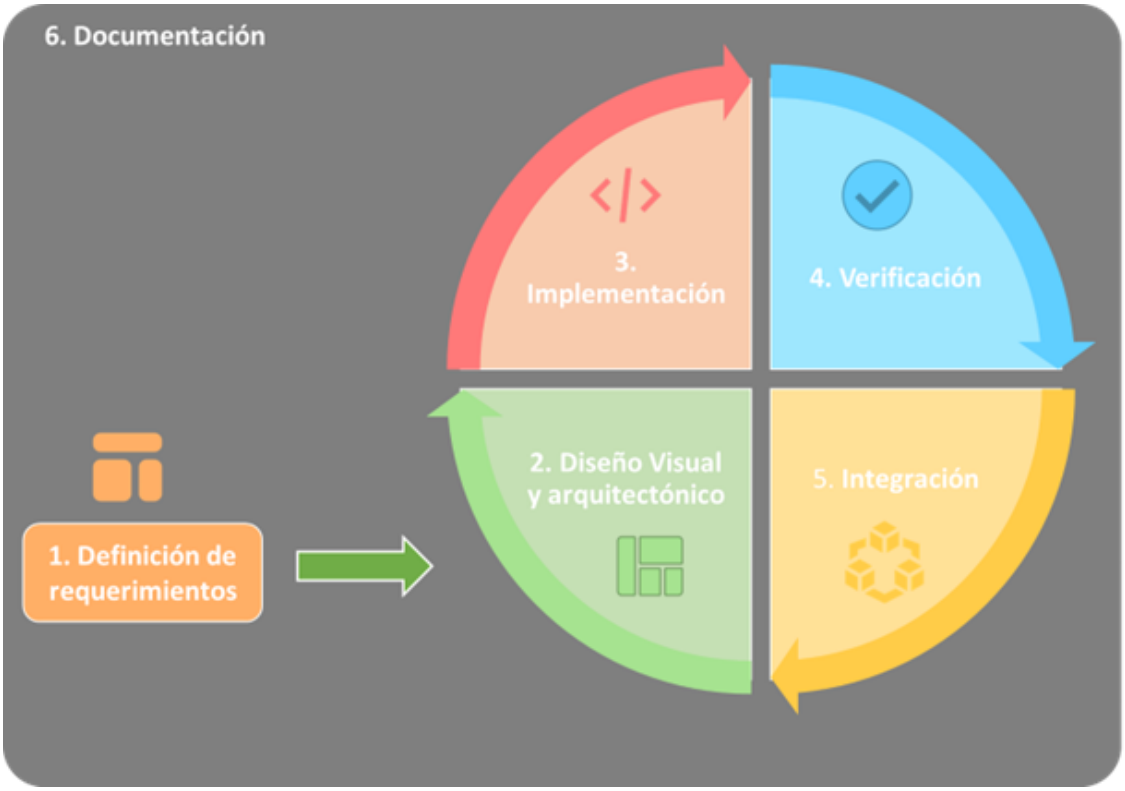

Figura 2. Diagrama de actividades

4.1. Definición de requerimientos

En este primer capítulo de la metodología, se ha centrado en la identificación y establecimiento de los requisitos necesarios para el desarrollo de la aplicación. El nicho de la aplicación está

orientada a un usuario en un concepto de administrativo con conocimientos en análisis de datos e información, con base en esto, los requisitos se utilizarán como guía en las etapas posteriores del desarrollo, tanto en aspectos visuales como funcionales, a continuación, se presentan los requisitos funcionales y no funcionales definidos para este proyecto.

4.1.1. Requerimientos Funcionales.

- La aplicación debe almacenar y obtener las configuraciones del usuario, como lo sería: su inicio de sesión, el tema escogido (oscuro o claro), el estado del panel de control, la información de cada bloque del panel (dispositivo, fechas y plantilla) y las plantillas pertenecientes al usuario.
- Las plantillas generadas por el usuario son únicamente pertenecientes a él mismo.
- Los datos de los dispositivos se deben obtener por un rango de fechas que puede variar de formato, pero en general, serán abarcados por un rango inicial hasta un último dato registrado que esté en tiempo real.
- Los datos que envíe un dispositivo en tiempo real deben ser recibidos por el *Backend*, que se encargará de almacenarlos y además, también será informado todo usuario que se encuentre suscrito al tópico.
- Cada una de las plantillas debe contar con opciones de personalización que permitan cambiar las propiedades de una gráfica como: tipo (forma), títulos de los ejes, colores y etiquetas de la gráfica.

4.1.2. Requerimientos No Funcionales.

- La aplicación debe contar con un *efecto de carga* que se debe mostrar siempre de los cabeceros de la ventana cuando se realiza alguna petición de recursos.
- Se debe proveer al usuario una guía rápida a manera de tutorial, haciéndole entender el funcionamiento básico de la aplicación.
- El usuario tendrá la disponibilidad de escoger entre un tema claro u oscuro para la vista de la plataforma, con el propósito de evitar la fatiga visual del mismo; la configuración se guardará en el navegador, por ende, cada vez que el usuario ingrese por el mismo navegador se cargará el tema previamente seleccionado, cabe resaltar que el tema por defecto será el tema claro.

4.1.3. Análisis general de Smart Campus UIS. Antes de definir el software a utilizar en la aplicación, era fundamental llevar a cabo una investigación de las características y limitaciones de la arquitectura Smart Campus UIS, como el sistema de autenticación, APIs y la gestión de los datos de los dispositivos. La figura 3 proporciona una representación visual del flujo de comunicación entre el *Backend* de gateway y el *Backend* de data en la arquitectura.

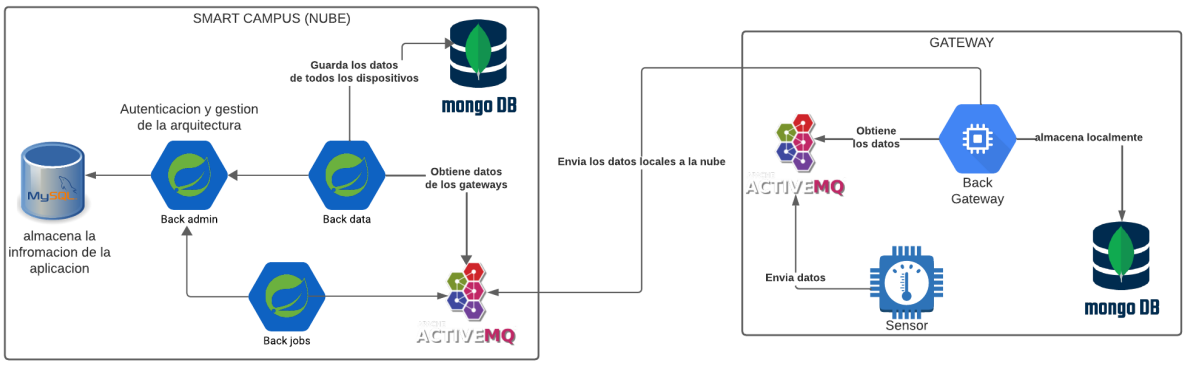

Figura 3. Comunicación Gateway y Backend de la arquitectura

4.1.3.1. Despliegue local. El proceso de investigación se inició con la implementación local de los componentes del *backend*, los cuales se mencionaron en el capítulo3.3. El propósito detrás de esto era comprender a detalle su funcionamiento técnico, durante este proceso, se identificó que los tres microservicios cumplían con los requisitos de un sistema distribuido, si bien, su ejecución se tornaba bastante compleja debido a su estricta dependencia mutua, esto imponía la obligación de ejecutar los servicios en un determinado orden para asegurar su correcto funcionamiento.

Sin embargo, el desafío más significativo surgió con la versión de *Java* y las propiedades de configuración de *Spring Boot*. Los microservicios fueron desarrollados en la versión 1.8 de *Java* del año 2016, versión que en la actualidad se muestra incompatible con varias bibliotecas de *Java*, utilizadas en el proyecto; para el caso de las propiedades de configuración, estas incluían referencias a las direcciones IP de los microservicios, las base de datos y el bróker de mensajería. El problema de modificar las propiedades, es que, requeriría realizar cambios al código fuente, un

ejemplo de esto, cambios en la dirección IP de la máquina que ejecutase un componente, requeriría cambiar el código fuente y recompilar la aplicación antes de poder hacer nuevamente funcional la arquitectura.

4.1.3.2. Obtener los datos de los dispositivos. Posteriormente, se procedió a analizar el proceso de obtención de datos suministrados por un dispositivo, aspecto crucial y de mayor interés en este proyecto, la obtención de esta información estaba condicionada a la necesidad de conocer: el Gateway al que está vinculado el dispositivo, como también el proceso específico al que pertenece dicho dispositivo. Los procesos son creados para un dispositivo al momento de ser registrados en el Gateway y cumplen principalmente la función de identificador de él, esta particularidad representa una redundancia en su uso, ya que, un proceso se comporta de manera análoga a un dispositivo. Para explicar de manera más conceptual, la figura 4 muestra el flujo de registro de un proceso y un dispositivo, recordando que un proceso también puede ser una aplicación de *Java*, como fue mencionado en el capítulo 3.2 con la intención de resaltar las similitudes entre procesos y dispositivos.

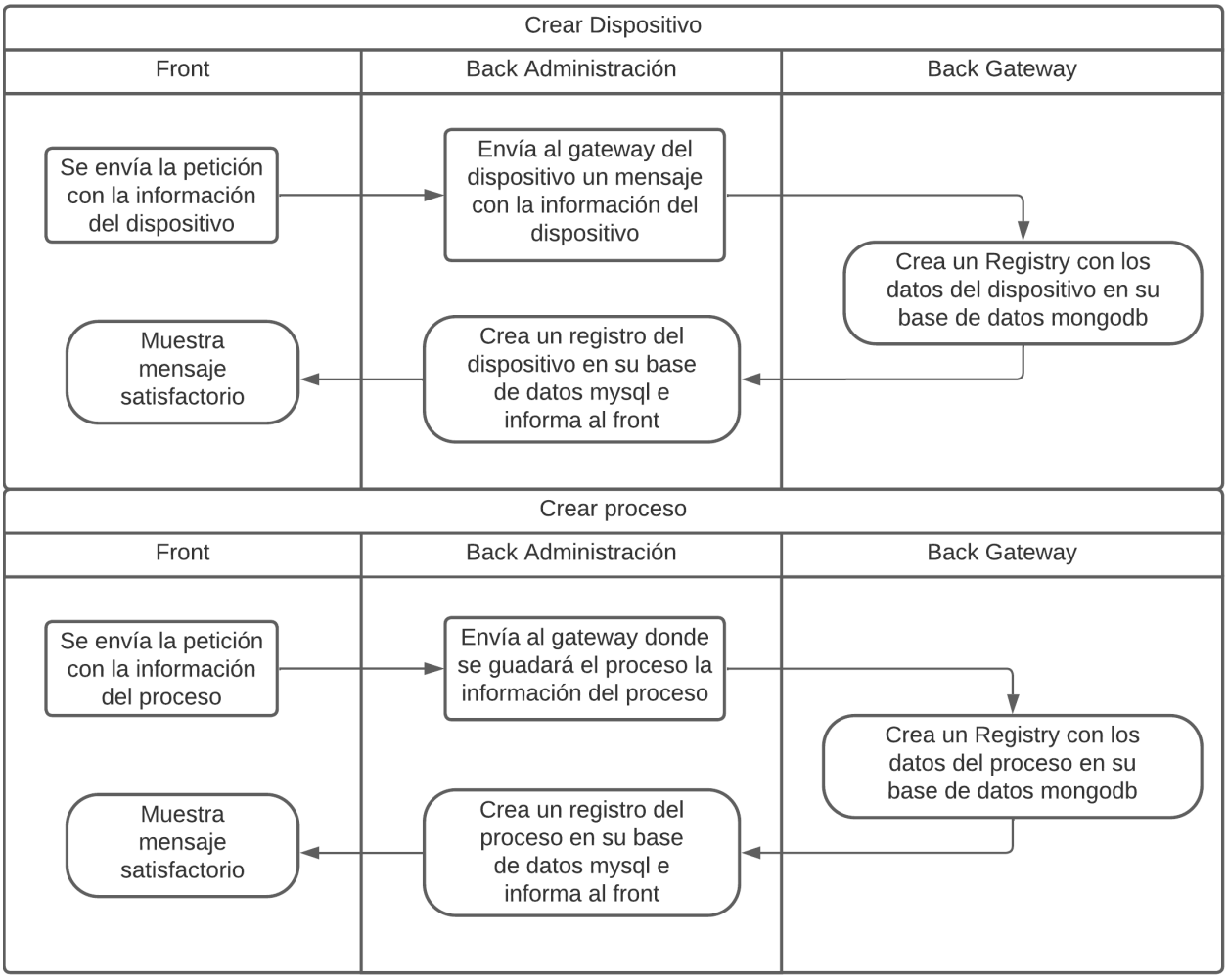

Figura 4. Flujo de creación de procesos y dispositivos

4.1.3.3. Evolución de los Backend de Administración y datos. En el momento de la realización de este proyecto se encontró que existía una segunda versión de la arquitectura, que busca abordar cierta mejora en el manejo de los dispositivos, en ella se busca simplificar la forma en que se gestionan, para esto, se ha prescindido del concepto de “procesos”, ya que, esta no era realmente necesaria. En términos simples, un dispositivo era un sensor o un actuador y un proceso

era una actividad en el Gateway que podía enviar información como un sensor o realizar una acción como actuador. La figura 5 representa el diagrama relacional planteado hasta el momento un dispositivo en esta segunda versión.

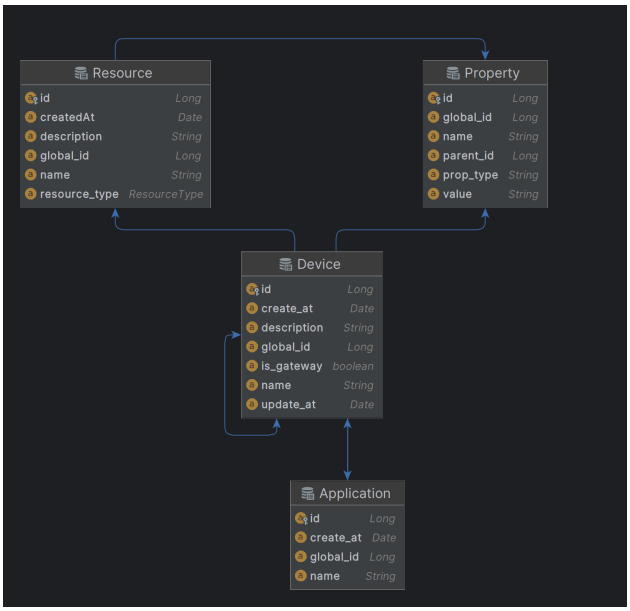

Figura 5. Diagrama relacional de los dispositivos segunda versión

4.1.3.4. Conclusión del análisis de la arquitectura. Como se ha mencionado hasta el momento, se identificaron dos versiones de la arquitectura, cada una con sus características. Como paso final de la investigación, se procedió a evaluar la versión con la que se iba a trabajar. Para tomar esta decisión, se realizó una comparación entre ambas versiones, con un enfoque en sencillez de la comunicación y obtención de datos de los dispositivos.

La primera versión de la arquitectura presentaba ciertos inconvenientes, como se mencionó previamente, relacionados con problemas en el despliegue y el uso de una versión obsoleta de Java, por lo tanto, se optó por trabajar con la segunda versión, con la intención de aprovechar su enfoque

simplificado en la obtención de datos de dispositivos, este enfoque se considera más adecuado para futuros desarrollos de la arquitectura.

4.2. Diseño visual y arquitectónico

Esta fase se centró en la creación de la parte visual, priorizando la comodidad del usuario como también el enfoque funcional de la aplicación; toda vista se planteó como una interfaz adaptable entre un tema claro y uno oscuro, esto mejora la accesibilidad a los usuarios en entornos de poca luz, y contribuye a reducir la fatiga visual. En última instancia, la inclusión de temas claros y oscuros en las interfaces es esencial para mejorar la experiencia del usuario y asegurar que la aplicación sea accesible y cómoda en diversas situaciones y para diferentes usuarios.

4.2.1. Prototipo visual. Para cualquier desarrollo de una aplicación con interfaz visual, resulta esencial diseñar un modelo de referencia a partir del cual, se construyó la interfaz comúnmente conocida como *"mockup"* en inglés. Por lo tanto, antes de adentrarse en la programación, se realizó una investigación para definir la herramienta que permitiría hacer un maquetado visual de las vistas con las que contaría el usuario, en este sentido, se optó por utilizar Figma, un editor de gráficos vectorizados diseñado para aplicaciones web. Figma destaca por sus complementos que ayudan a agilizar el flujo de trabajo, además de contar con una capa gratuita de uso.

Cabe destacar que estos diseños, posterior a pasar por la etapa de verificación 4.4, y la de integración 4.5, pueden sufrir mejoras o ajustes en su estructura debido a diversos factores. Estos

factores pueden incluir un diseño que no resulte intuitivo ni óptimo para el usuario, la necesidad de incorporar elementos adicionales como: botones, cambios en la paleta de colores, modificaciones derivadas de cambios en el software u otros elementos relacionados.

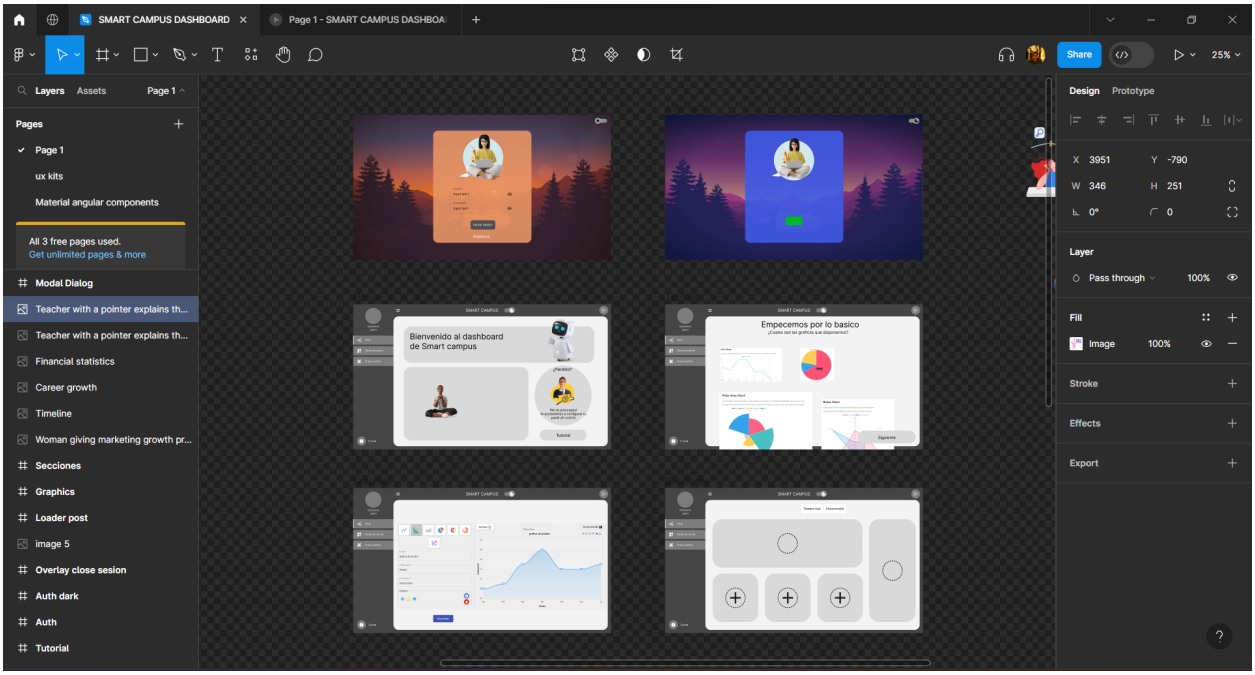

Figura 6. Interfaz de Figma e ilustración de la aplicación

En Figma, se diseñaron los prototipos visuales de toda la aplicación, utilizando iconos y componentes disponibles en Material *Angular*, el objetivo de esto, era garantizar la menor diferencia posible entre el diseño y la implementación.

A continuación, se presentarán las vistas de la aplicación y se proporcionará una breve descripción funcional, destacando como el usuario interactúa con ellas.

4.2.1.1. Inicio de sesión del usuario. La primera vista que podrá visualizar un usuario sin autenticar es la de autenticación, en caso contrario sería redirigido a la vista 4.2.1.2. Este diseño también busca ser útil para los próximos sistemas, ya que puede ser reutilizado y ser la vista de autenticación de todos los módulos añadidos a la arquitectura. sin embargo, como se habló en el capítulo 4.1.3, se va a trabajar sobre la segunda versión de la arquitectura, la cual, no cuenta con el sistema de autenticación, esto hizo que fuera necesario implementar la interfaz para esta parte, sin embargo, se hizo el diseño para su implementación futura de ser posible.

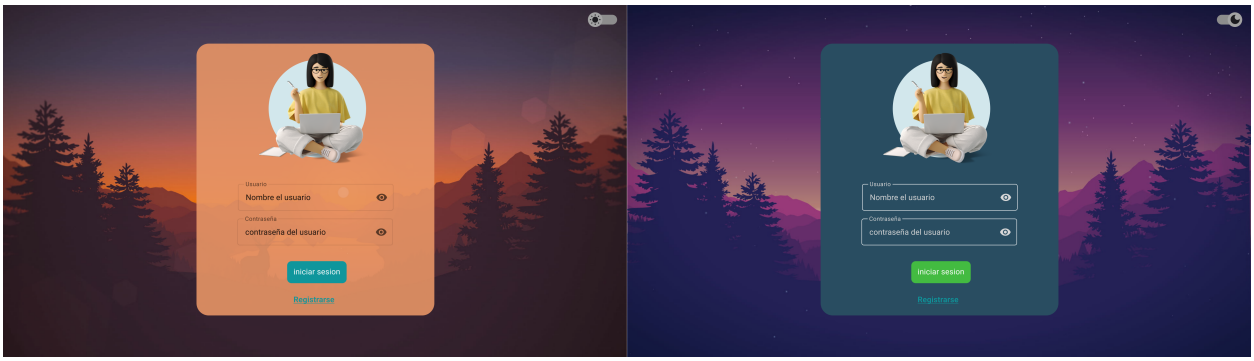

Figura 7. Ilustración del inicio de sesión

4.2.1.2. Inicio del usuario. Esta vista es la primera interfaz que el usuario experimentará al acceder a la aplicación, accesible a través del menú lateral izquierdo. En la creación de esta vista, se ha tenido en cuenta que no podía estar cargada de mucha información, ya que a menudo pueden pasar por alto esta sección. Como consecuencia de esto, se decidió realizar un diseño minimalista, que permitiera al usuario ver de manera rápida y concisa la información más relevante, como lo es el nombre de la aplicación, una breve descripción de su naturaleza y utilidad, junto a un botón de ayuda o tutorial.

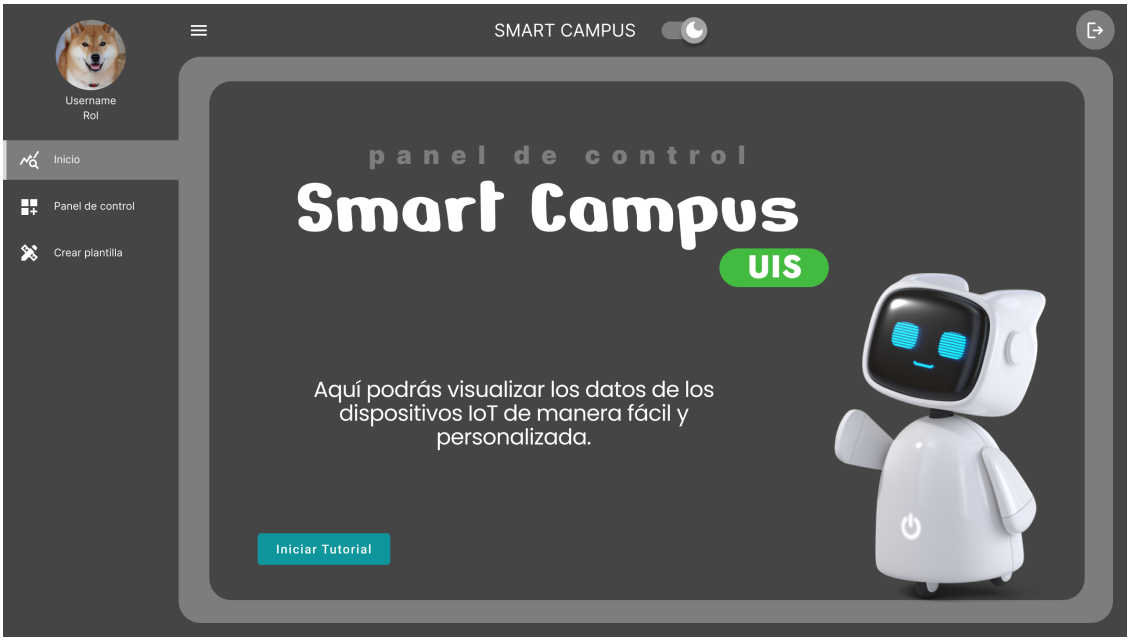

Figura 8. Ilustración del inicio de usuario

4.2.1.3. Tutorial. Esta es la vista del tutorial que proporciona una guía para familiarizar a los usuarios con las capacidades y herramientas clave de la aplicación, lo que facilita una experiencia de usuario más efectiva y productiva. la cual consta de tres secciones, que se mencionaran a continuación.

En la primera sección, se aborda la temática de los dispositivos, se explica cómo se pueden visualizar los datos de los mismos en forma de gráficas, y se presenta una descripción concisa de los tipos de gráficas específicos utilizados en este proyecto.

La segunda sección del tutorial se centra en el concepto de plantillas y ofrece una visión general de las opciones disponibles para personalizar estas plantillas según las preferencias del usuario.

La tercera y última sección del tutorial se dedica a explicar el proceso de creación de una gráfica a partir de los datos proporcionados por un sensor en el panel de control del usuario.

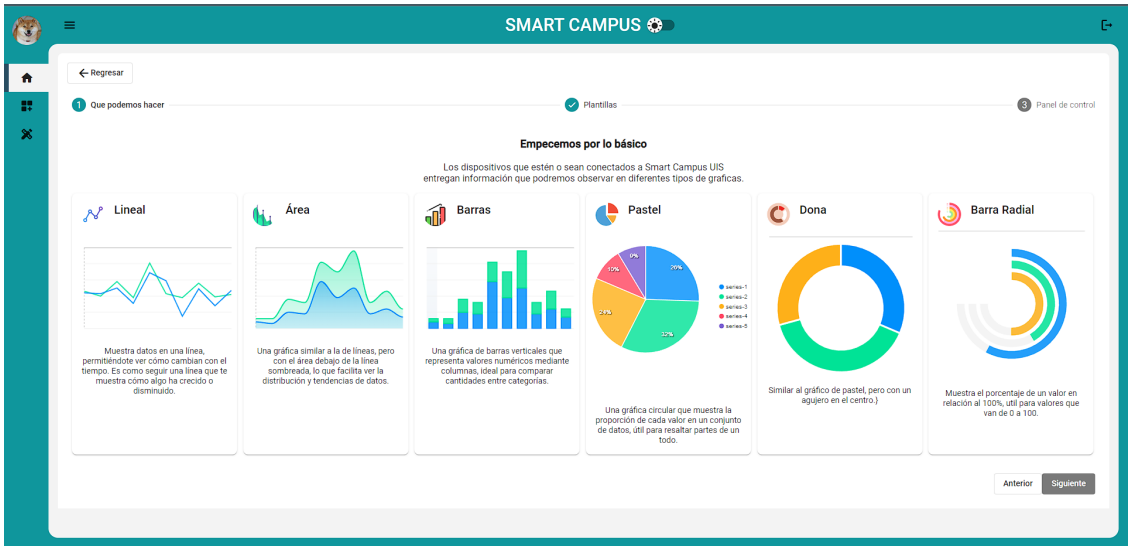

Figura 9. Ilustración del tutorial

4.2.1.4. Menú panel de control. El segundo menú, denomina *"Panel de control"*, desempeña un papel fundamental en la aplicación, ya que permite a los usuarios crear su propio panel de control personalizado. En este menú existen dos conceptos clave: "bloque" y "gráfica". Los "bloques" se refieren a las tarjetas en las que se pueden ver las gráficas. Estos bloques son altamente personalizables, lo que incluye la capacidad de ajustar su tamaño y posición dentro de un área específica del panel de control. Los bloques también se pueden eliminar según las preferencias del usuario.

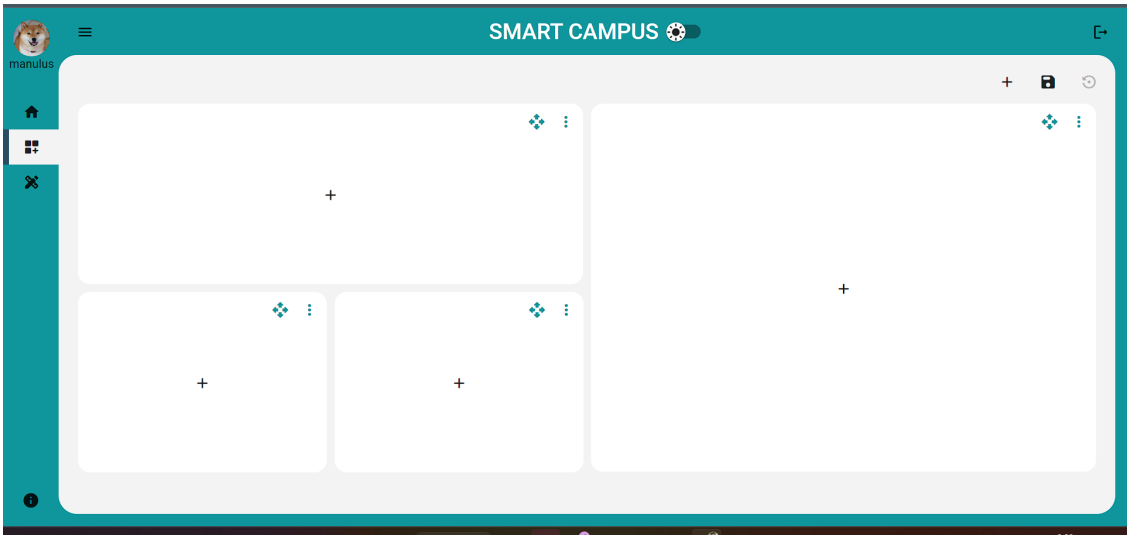

Figura 10. Menú panel de control

En esta vista se incluye una ventana emergente que permite configurar la información necesaria para crear y mostrar una gráfica en un bloque específico. Aquí es importante destacar que si es selecciona el modo de visualización en tiempo real, no se puede seleccionar una fecha de finalización. Esto porque para este caso se centra únicamente en mostrar los datos desde la fecha de inicio, hasta el dato más reciente publicado por el dispositivo.

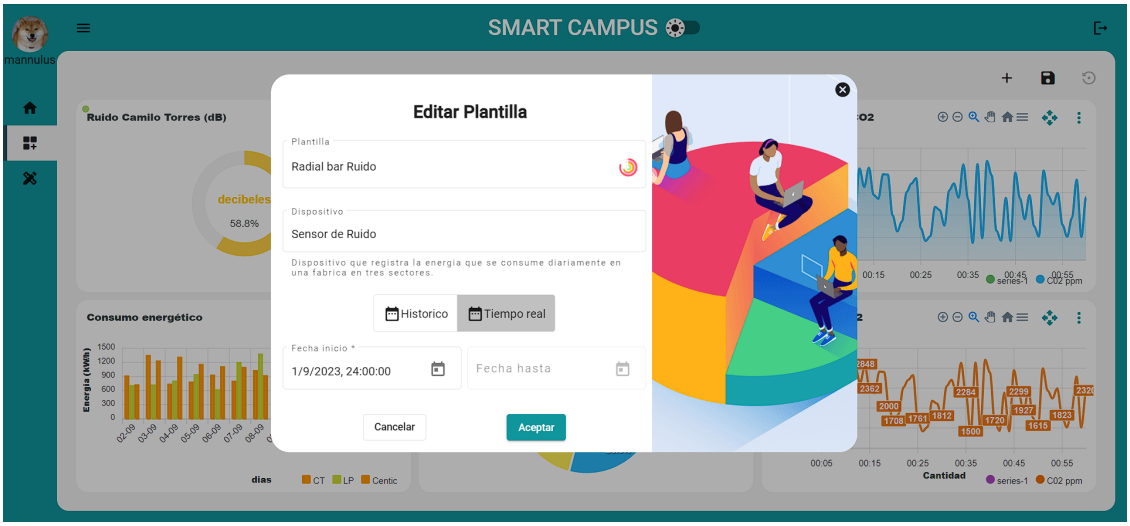

Figura 11. Vista panel de control

4.2.1.5. Crear plantilla. Aquí encontraremos una interfaz dedicada a diseñar las plantillas que servirán como base para la creación de las gráficas. Como se apreció en la ventana emergente de la figura 11, uno de los campos permite a los usuarios seleccionar entre diferentes plantillas disponibles. En el capitulo 4.3

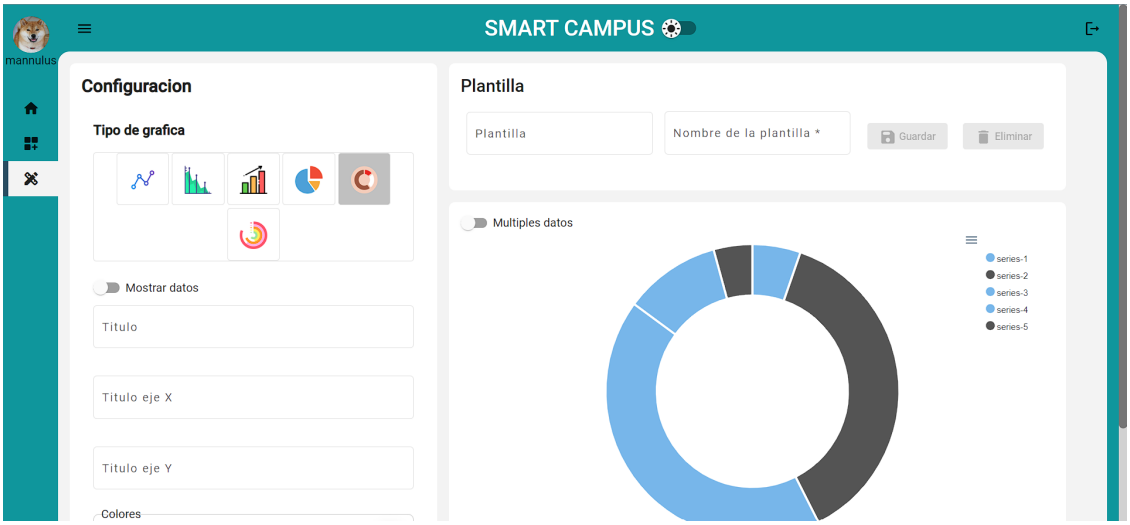

Figura 12. Vista creador de plantillas

4.2.2. Definición del software. Se optó por desarrollar una aplicación web debido a su gran popularidad y escalabilidad. Como Framework de desarrollo web se utilizó *Angular*, el cual dispone de numerosas dependencias que facilitan el desarrollo y cuenta con una documentación sólida. La versión utilizada fue la 16.1.6, la cual era la versión más reciente y estable, disponible en el momento de la ejecución de este proyecto.

4.2.2.1. Librerías para las gráficas. En esta etapa del diseño del proyecto, se planteó el uso de una biblioteca de gráficos que permitiera la creación de diferentes tipos de gráficas altamente personalizables, de código abierto y con la condición de ser compatible con *Angular*, esto porque es un gran inconveniente utilizar librerías que no concuerden con la estructura lógica de *Angular*, por ende, sería un problema para el desarrollo a largo plazo, ya que, la persona que retome o desee modificar la aplicación, le debe ser sencillo aprender una herramienta con un mecanismo común de funcionamiento, esto para facilitar la continuación del proyecto como también una norma básica para mantener un estándar al tratar con Frameworks.

Para llevar a cabo este proceso de selección se realizó una búsqueda donde se identificaron tres opciones potenciales: *Python dash, Chart.js* y *ApexChart*. Para tomar una decisión informada, se procedió a hacer una prueba de integración con *Angular* y así evidenciar ventajas y desventajas del uso de cada una. A continuación se detallarán las características y ventajas de cada opción:

- *Python dash* es una librería de código abierto que se utiliza para crear paneles de control de datos de manera sencilla y eficiente. Se desarrolla en Python y se basa en bibliotecas

populares como Plotly y Flask. Ahora, aunque Python es un lenguaje bastante usado para el tratamiento de datos, siendo esto una gran ventaja, en una prueba de integración se observó que, al utilizar Flask, era necesario ejecutarlo como un servidor y para luego incrustarlo en el HTML de la aplicación mediante una etiqueta iframe. Esto introduce una arquitectura MVC (Modelo Vista Controlador), lo que implica solicitar cambios al servidor en la vista y esperar a que este procesa la información antes de devolver una vista al navegador, esto demostró ser una desventaja, ya que no permite tener control directo de la gráfica, perdiendo la característica de ser personalizable.

- *Chart.js* es una librería *JavaScript* de código abierto diseñada para crear visualizaciones de datos interactivas y gráficos en páginas web. Estas gráficas se construyen en un lienzo (canvas) y son altamente personalizables. Aunque es compatible con *Angular* gracias a la dependencia ng2-charts, es una herramienta externa que presenta desafíos en su integración con angular, en determinadas ocasiones los componentes al cambiar sus propiedades en tiempo de ejecución, *Angular* no detectaba los cambios en el componente y, por lo tanto, no permitía renderizar los cambios de la vista.

- *ApexChart* es una librería de *JavaScript* basada en *Chart.js*, la cual ofrece un alto grado de personalización de prácticamente todos los aspectos de los gráficos, incluyendo colores, etiquetas, estilos, animaciones y marcadores, lo que permite adaptar las visualizaciones a nuestras necesidades específicas. Los gráficos generados son altamente interactivos, con herramientas como; resaltar datos, ver detalles emergentes (tooltips), hacer Zoom, descargar

una imagen de la gráfica en formato PNG o SVG, descargar los datos en formato CSV y entre otras características. Es compatible de manera nativa con *Angular*, *React* y *Vue* y su documentación cuenta con ejemplos de uso para cada Framework.

Tras esta investigación y pruebas de integración realizadas, se tomó la decisión de utilizar *ApexChart*. Esta decisión es basada en su capacidad para cumplir con los requisitos funcionales del proyecto, cuenta con aspectos positivos como su facilidad de configuración, la capacidad de personalizar estilos (temas) y su estructura de datos simple, además, la integración con *Angular* y *TypeScript* resultó ser completa, y al basarse en la reconocida librería *Chart.js*, se aseguró un soporte a largo plazo.

4.2.3. Tipos de gráficas utilizadas. En línea con lo discutido en el capítulo anterior, /textit*ApexChart* proporciona una amplia variedad de tipos de gráficos. Para este proyecto, se seleccionaron seis tipos de gráficos que se consideraron los más relevantes y adecuados para los casos de uso planteados en la fase de integración 4.5

4.2.3.1. Gráfica lineal o de área. Este tipo de gráfico se utiliza para representar las tendencias y cambios de los datos a lo largo del tiempo. La representación de áreas bajo la curva en un gráfico de área agrega un componente visual adicional para resaltar la magnitud de los valores. Esta visualización se eligió para casos en los que es crucial mostrar la evolución de los datos a lo largo del tiempo, como en la monitorización de sensores a lo largo de días, semanas o meses.

4.2.3.2. Gráfico de barras. El gráfico de barras es una representación efectiva para comparar diferentes categorías o conjuntos de datos. Se utilizan barras verticales para mostrar los valores numéricos de determinadas categorías. Este tipo de gráfico se eligió para situaciones en las que es necesario comparar rápidamente datos entre diferentes categorías o elementos, como la comparación de datos de temperatura entre varios dispositivos.

4.2.3.3. Gráfico de pastel o dona. El gráfico de pastel o dona se utiliza para mostrar la proporción de cada elemento en un conjunto de datos en relación con el todo. Es ideal para representar datos de manera que sea fácil de entender la contribución de cada elemento al conjunto total. Este tipo de gráfico se utiliza cuando se desea mostrar el porcentaje de contribución de los datos de cada conjunto.

4.2.3.4. Gráfico de barra radial. La barra radial es un gráfico circular que se utiliza para representar datos en categorías con valores porcentuales. Este tipo de gráfico se eligió para casos en los que es importante mostrar datos en un formato porcentual a un círculo, como el porcentaje de llenado de un tanque de agua.

4.2.4. Estructura del proyecto. Como fue menciono en el capítulo 4.2.2, el proyecto se desarrolló en *Angular*, y por esta razón la estructura de carpetas del proyecto se basó, en la recomendada del Framework. Estas buenas prácticas permitieron la creación de componentes reutilizables que evitaron la duplicación de código y que a su vez agilizan el desarrollo. La figura 13 explica esta estructura.

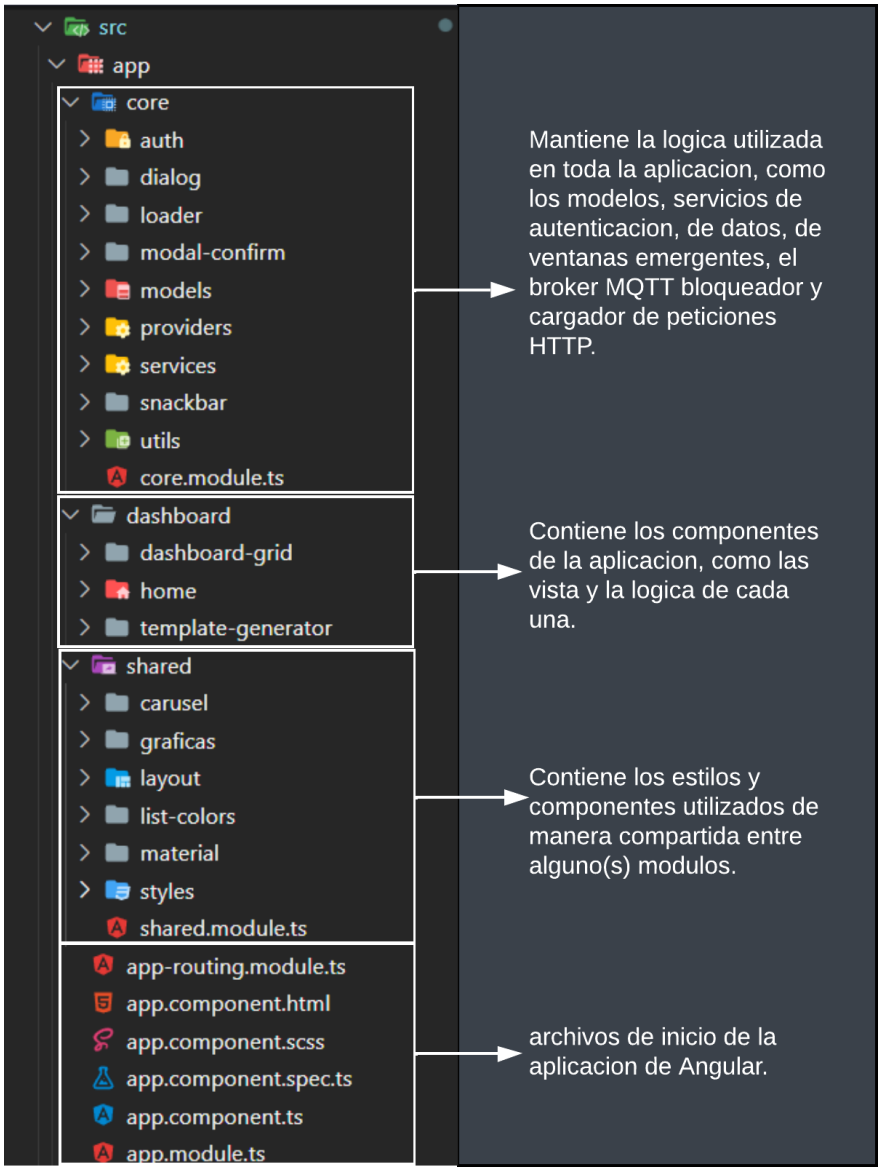

Figura 13. Estructura de carpetas del proyecto

La carpeta *core* es responsable de mantener los servicios, modelos, interceptores, guardianes y demás dependencias globales que son utilizadas en todo el proyecto. Entre los elementos clave en esta carpeta se encuentran:

- Se crearon varios servicios para la gestión de diversas funcionalidades, como un creador de ventanas emergentes con un estilo global, que simplifica la creación de ventanas de confirmación y de formularios. También se implementaron servicios para interactuar con la API del *Backend* de datos y de administración, junto con sus modelos de datos correspondientes.

- Se implementó un interceptor de peticiones HTTP utilizado para asignarle el ID del usuario a la cabecera de las solicitudes y crear un *bloqueo* para peticiones de tipo POST y un cargador para peticiones de tipo GET.

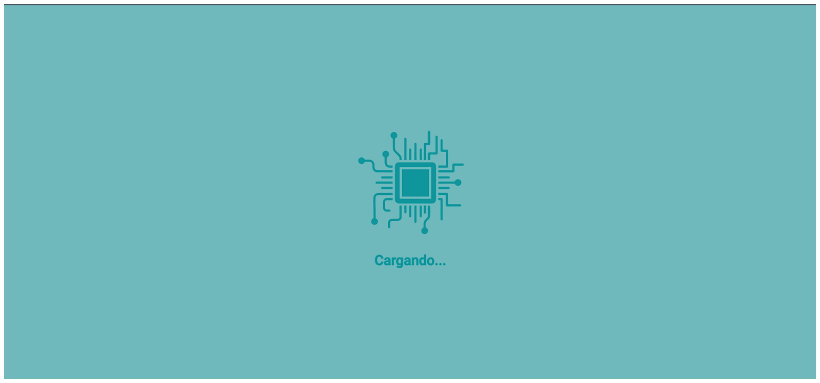

Figura 14. Bloqueador de peticiones HTTP tipo POST

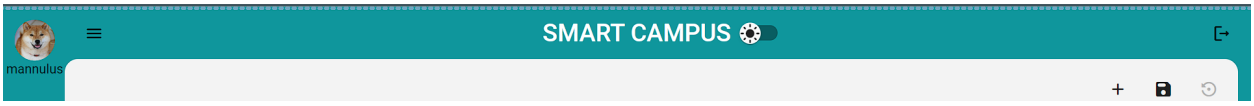

Figura 15. Cargador de peticiones HTTP tipo GET

4.3. Implementación

En esta fase, se desarrolló toda la lógica de la aplicación, las vistas y los arreglos necesarios al *Backend* de administración y datos, para que cumplieran con los requerimientos del capítulo 4.1

y diseños visuales realizados en 4.2. Es importante aclarar que en ciertos momentos se tomaron decisiones creativas en cuanto al diseño visual, pero siempre se procuró mantener actualizadas las vistas de Figma, con el diseño implementado.

4.3.1. Creador de plantillas. El primer módulo de donde parte el desarrollado, es el sistema para crear plantillas, gracias a los ejemplos de la documentación de *ApexChart* se logró definir un esquema de comunicación entre el formulario de personalización y el modelo de configuración de las gráficas. Este formulario guarda una configuración que se le denomina plantilla, con la cual el usuario puede tener una base a utilizar en el panel de control, donde con cada diseño que haya realizado creara gráficas dentro de los bloques. Esta vista consta de varias partes que a continuación se entrara en detalle.

El primer formulario, titulado como “Configuración”, permite a los usuarios definir diversos aspectos de la gráfica, como el tipo de gráfica, la visibilidad de las etiquetas, la personalización de los títulos horizontal y vertical, la cabecera y la capacidad de ajustar los colores de la gráfica.

El segundo formulario ofrece la posibilidad de seleccionar una plantilla previamente creada o crear una nueva plantilla a la que pueden asignar un nombre. A su vez se incluye la opción de eliminar plantillas existentes.

Por último, en la interfaz se muestra una vista previa de la gráfica que se generará en función de todas las configuraciones previamente mencionadas y unos datos generados aleatorios para cada tipo de gráfica. Esto funciona como una previsualización del cómo será la gráfica.

La figura 16 proporciona una ilustración que aclara la relación de cada campo del formulario con su efecto en la gráfica. Es importante destacar que en algunos casos, una gráfica puede contener múltiples líneas, barras o segmentos circulares, y es por esta razón, que se permite la selección de varios colores en el formulario, los cuales se aplican a las gráficas en el mismo orden en el que se encuentren.

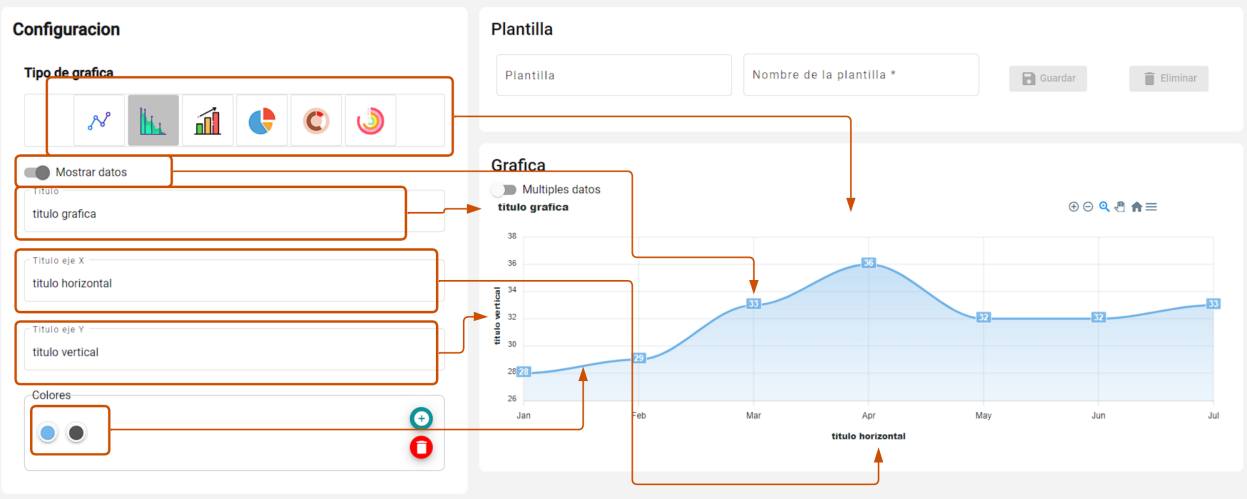

Figura 16. Vista formulario de plantillas

4.3.2. Panel de control. El panel de control, es la vista principal de la aplicación, diseñada para crear bloques asociados a plantillas y dispositivos. para los usuarios nuevos la vista que es mostrada, es una distribución predeterminada de los bloques, en caso contrario, siempre y cuando el usuario posee una distribución, esta será cargada. Los bloques son creados en una cuadrícula adaptable de filas y columnas, facilitada por la biblioteca Gridster. Los bloques tienen la característica de poder ser reorganizados mediante el botón de “Mover”, que permite arrastrarlos y soltarlos en la ubicación deseada, también se pueden redimensionar desde las esquinas del bloque,

para ajustar su tamaño, siempre manteniéndolos visibles en la ventana sin necesidad de crear una barra de desplazamiento (scroll). Cada bloque cuenta con un botón para eliminarlo, el cual muestra un mensaje de confirmación cuando se intenta borrar un bloque que ya contiene una gráfica.

En la parte superior de la pantalla, se encuentran las acciones principales, como crear, guardar y descartar. El botón de “Agregar Bloque” ubicado en la parte superior permite agregar nuevos bloques a la cuadrícula y la ubicación de estos bloques es gestionada por la librería, que busca llenar los espacios vacíos o agregarlos en nuevas filas o columnas, ajustando automáticamente la vista.

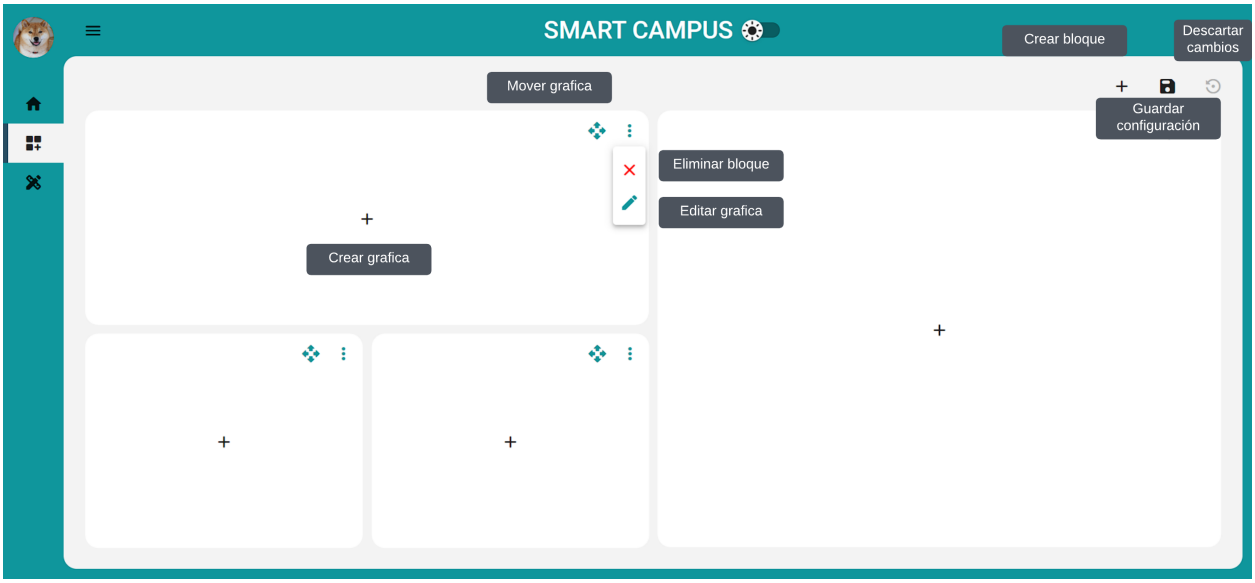

Figura 17. Tooltips panel de control

Es importante mencionar que aunque la vista no cuenta con un sistema de guardado automático, sí con una ventana emergente preventiva, haciendo que el usuario no pueda navegar entre menús a menos que guarde o descarte los cambios. Para guardar los cambios o restaurar la vista

a la última versión guardada, el usuario deben utilizar los botones correspondientes. El botón de “Restaurar” solo se habilita cuando se ha realizado alguna modificación en la vista.

4.3.2.1. Asociar plantillas y gráficas a un bloque. Dentro de un bloque, al ser presionar el icono con un mas o el icono de edición con un lápiz, se abre una ventana emergente que permite configurar la información necesaria para crear y mostrar una gráfica en un bloque específico. Al realizar cambios y guardarlos en esta ventana, se muestra un icono de carga en el bloque hasta que la información del dispositivo se obtenga con éxito y se muestre la gráfica correspondiente. Para obtener estos datos, se realiza una petición al *Backend* de datos, en caso de seleccionar el modo "tiempo real", se envía otra petición para obtener una cadena de texto que contiene el nombre del tópico al cual el dispositivo envía información. Este tópico es el que contiene exclusivamente los mensajes enviados por ese dispositivo, a diferencia de “device-messages”, que recibe los datos de todos los dispositivos.

Es relevante considerar el rango de fechas establecido en la ventana de configuración, porque, para algunos sensores, su comportamiento es enviar información cada segundo o cada minuto, por lo que en un rango de días, la cantidad de datos puede ser considerable, lo que podría causar un impacto en el rendimiento tanto del *Backend* como del frontend.

4.3.3. Componente de campo autocompletado para listas. Este componente se creó para facilitar una lista desplegable en un campo de tipo selector, pero con la particularidad de que se puede buscar. Aunque actualmente material *Angular* ya posee una característica similar,

para este caso se modificó para que al escribir se permitieran peticiones de búsqueda al servidor, donde se le indica que se desea buscar en esa lista, todas las coincidencias con el texto ingresado.

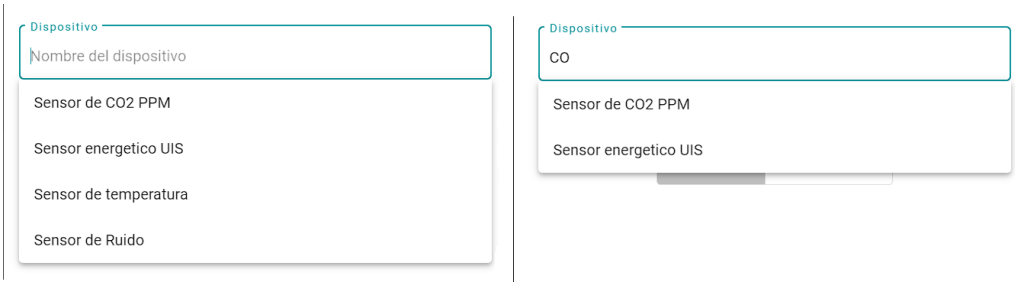

Figura 18. Componente auto completado asincrónico de listas

4.3.4. Sistema de integración continua CI/CD. Para facilitar la comunicación con los directores de este proyecto, se implementó un sistema de integración continúa utilizando GitHub Actions, plataforma que permite automatizar tareas como la compilación, las pruebas e implementación de la aplicación, la cual es gratis con una cuota de 2000 minutos de ejecución mensual (Github, 2023), lo que resulta más que suficiente para los requerimientos del proyecto. Esto se realizó para el back de administración, datos y el Front de esta proyecto. Resulto bastante beneficio al momento de desarrollar, ya que permitió reducir la necesidad de mantener múltiples aplicaciones en funcionamiento simultáneamente, para evitar sobrecargar los recursos de la máquina local.

4.4. Verificación

La fase de verificación fue un paso intermedio entre la implementación y la integración, el cual su objetivo fue hacer una validación del proceso realizado hasta el momento, con el pro-

pósito de verificar el cumplimiento de los objetivos establecidos, los requerimientos y el diseño visual y arquitectónico. Esta etapa además permitió identificar factores que pudieran haber quedado pendientes o no se hubieran tenido en cuenta, para ser corregidos y complementar las tres fases anteriores.

Hasta este punto de la metodología, se ha evidenciado el cumplimiento exitoso de los primeros tres objetivos específicos. En primer lugar, se logro identificar la necesidad de ofrecer una variedad de gráficas que permitieran a los usuarios visualizar y comprender los datos de manera efectiva, y para ello se definieron los seis tipos gráficas a utilizar en este proyecto. En segundo lugar, se diseñó un sistema de plantillas y un panel de control personalizable, que se adaptara a las necesidades individuales del usuario. Por último, se desarrolló un diseño o mockup que sirvió como base visual para la interfaz de la aplicación.

Con el último objetivo específico en mente, el siguiente capítulo presentará los escenarios de prueba planteados para los seis tipos de gráficas definidos en 4.2.3, buscando cumplir con el objetivo general y dar por terminado este proyecto.

4.5. Integración

En la fase de integración, se realizaron las pruebas para comprobar el funcionamiento de la aplicación cuando es conectada a la arquitectura Smart Campus UIS segunda versión. En este escenario, se dispuso de dispositivos que están conectados y enviando información en tiempo real para ser almacenada en el histórico de datos.

Para llevar a cabo estas pruebas, se tuvo en cuenta que los dispositivos no se podrían conectar a la arquitectura debido a que estamos trabajando sobre la versión 2.0, que carecía de la lógica necesaria para permitir una interacción completa entre el *backend* de datos y el *backend* de gateway. Por lo tanto, se tomó la decisión de implementar un programa que se hiciera pasar por dispositivo que se conecta con el bróker de la arquitectura, para enviar la información en los tópicos de "device-messagez "device-uuid"donde el uuid es identificador del dispositivo.

4.5.1. Aplicación de las pruebas. El programa se desarrolló utilizando tecnología web, como HTML y *JavaScript*, donde se emplearon librerías relevantes, como *paho MQtt* para las conexiones por WebSocket al bróker de *ActiveMQ* y *Faker.js* para generar valores aleatorios dentro de un rango específico, estos valores representan los datos que el sensor enviaría. La figura 19 presenta la interfaz que se diseñó para hacer las pruebas.

En esta interfaz, primero tenemos un formulario con campos para establecer la conexión con el bróker MQTT, los campos incluyen la dirección IP, el puerto, el tópico, el nombre de usuario y la contraseña. Una vez que se presiona el botón Çonectar", la librería Paho MQTT establece la conexión con el bróker utilizando la tecnología WebSocket. Esto nos permite simular la conexión de un dispositivo a la arquitectura de Smart Campus.

En esta interfaz como primero disponemos de un formulario con unos campos para conectarnos al bróker, como la IP, el puerto, el tópico, usuario y contraseña, luego de ser presionado el botón de *conectar*, la librería de paho MQTT realizara la conexión con el bróker mediante la

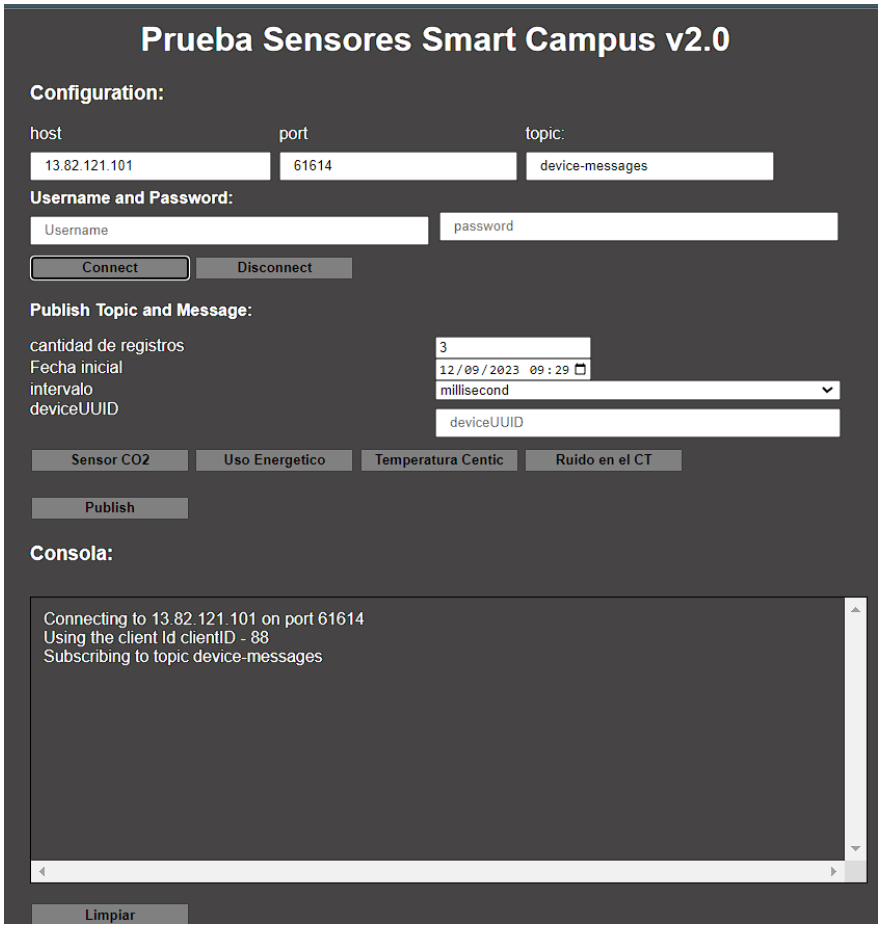

Figura 19. Interfaz de pruebas para los sensores

tecnología de WebSocket; Esto permitió simular el dispositivo conectado a la arquitectura.

Luego, encontramos campos que permiten configurar la cantidad de datos que se generaran, a partir de qué fecha, con qué frecuencia y desde qué dispositivo. Estos datos se envían al tópico "device-message"para que el *Backend* de datos los reciba y los almacene en la base de datos MongoDB. También se envían al tópico "device-uuid", donde el UUID es el identificador único del dispositivo, el frontend se conecta a este último tópico para escuchar cada vez que el dispositivo envía información, lo que le permite actualizar las gráficas configuradas para mostrar datos en

tiempo real de ese dispositivo.

Para probar esta herramienta y poder cumplir con el último objetivo específico de este proyecto, se propuso cuatro escenarios de prueba para sensores conectados a Smart Campus UIS.

4.5.2. Niveles de CO2 en el ambiente. Este caso de uso consistió en un sensor de calidad del aire, que monitoriza la cantidad de CO2 (dióxido de carbono) en el ambiente, con una unidad de medida que está dada en partes por millón (ppm). Estos datos se generaron en un rango que va desde 600 hasta 1400 ppm, con un intervalo de uno por cada segundo. Este rango se basó en los datos obtenidos en el proyecto de grado titulado *"Sistema de información basado en IoT para medición de la calidad del aire y seguimiento de rutas en sistemas de transporte público"* (Pérez & Claro, 2023) . La gráfica que se utilizó fue la lineal y la de área, ya que igual que en el proyecto mencionado, esta gráfica permite mostrar, a lo largo del tiempo, el comportamiento de los datos.

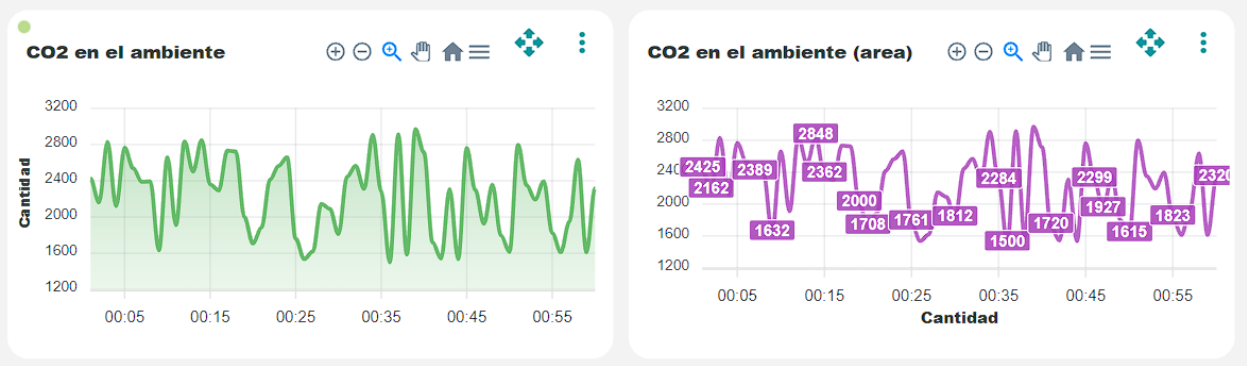

Figura 20. Caso de uso sensor energético

4.5.3. Sensor energético UIS. El sensor energético se encarga de medir diariamente el consumo total de energía entre tres edificios de la Universidad Industrial de Santander:

el Camilo Torres, el CENTIC y el edificio de Laboratorios Pesados. La gráfica seleccionada para representar este caso de uso fue la de barras, ya que por se asemejan con las gráficas que se encuentran en las facturas de la energía, que permite comparar los valores de consumo a lo largo del tiempo.

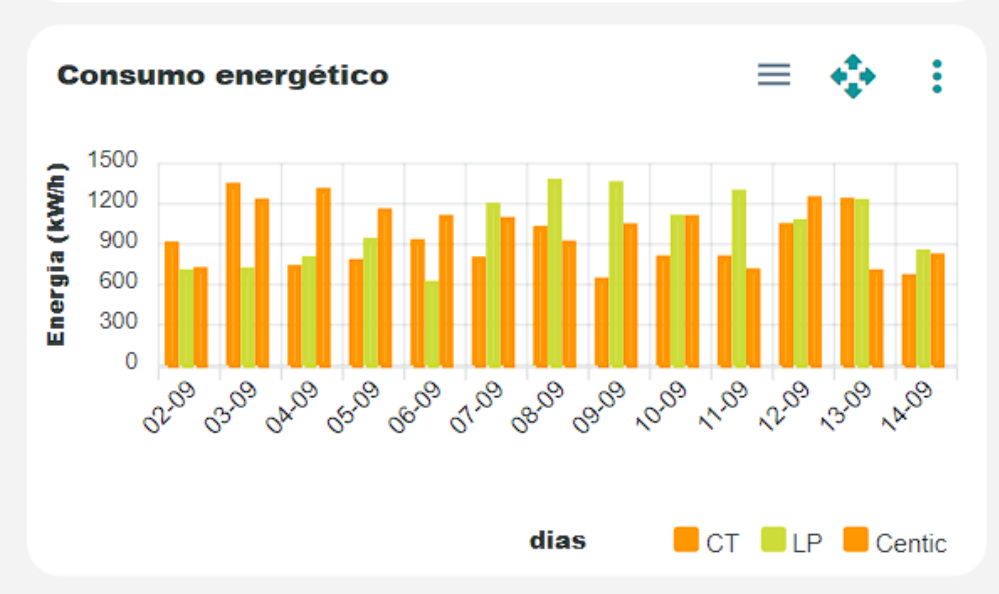

Figura 21. Caso de uso sensor energético

4.5.4. Sensor de préstamo de tarjetas del CENTIC. Este sensor registra diariamente la cantidad de tarjetas de acceso prestadas en el CENTIC. Se utiliza un sensor que lleva el seguimiento de las tarjetas prestadas, así, cada que es presta a una tarjeta al personal universitario, se registra como prestada y cuando la devuelve es registrada como no prestada, además de realizar un seguimiento de las tarjetas asignadas a administrativos del edificio. Al final del día, el sensor proporciona información valiosa al registrar el número de tarjetas prestadas, no prestadas y asignadas. Esto permite una gestión más efectiva de las tarjetas y contribuye a garantizar que estén

disponibles cuando se necesiten.

Figura 22. Caso de uso tarjetas CENTIC

4.5.5. Sensor de Ruido. Este caso de uso consiste en un sensor que mide el nivel de ruido en decibeles (dB), el sensor envía un número aleatorio entre 20 a 90 cada minuto. La gráfica que se escogió para proyectar estos datos fue una radial, ya que como se había mencionado en 4.2.3.4, esta gráfica permite promediar los datos y mostrarlos en una barra circular de 0 a 100. Esta gráfica es adecuada para este caso, ya que el umbral de normal de ruido está entre 20 a 100 decibeles, luego de eso el ruido es muy alto y, por lo tanto, perjudicial para el ser humano (RTVE.es, 2010). Para aclarar esto, la siguiente imagen muestra dos casos, uno cuando el ruido está alrededor de 58 dB y otro cuando está por encima de 100 dB.

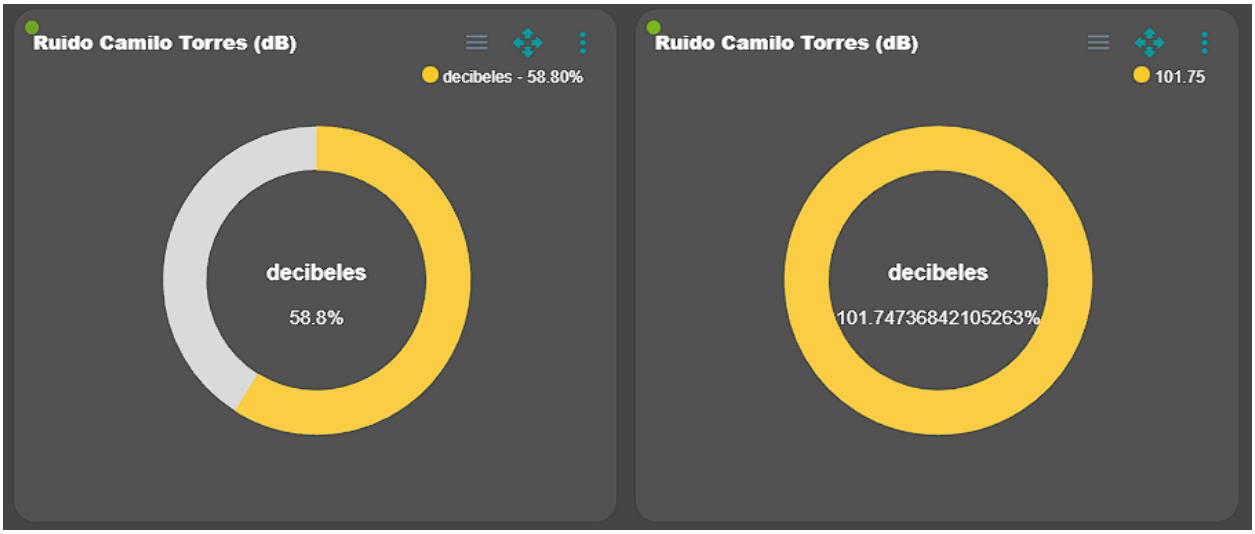

Figura 23. Caso de uso sensor de ruido

4.6. Documentación

Esta última fase de la metodología del proyecto, contó con la peculiaridad de estar presente de manera paralela durante todo el desarrollo del proyecto, esto se hizo con el propósito de evitar la realización de la documentación en el último momento y garantizar que las ideas y conceptos generados durante el desarrollo se reflejen adecuadamente en la documentación final. Con la culminación de esta fase, se dan por cumplidos los objetivos del proyecto.

5. Conclusiones

En armonía con los objetivos inicialmente establecidos, este proyecto se fundamentó en la necesidad de una aplicación que permitiera la visualización de datos provenientes de los dispositivos de Smart Campus UIS. Hasta ese momento. La aplicación web existente se centraba en tareas de administración, careciendo de la capacidad de presentar los datos generados por los dispositivos, sin embargo, se logró diseñar e implementar un panel de control que, a través de la creación de plantillas personalizables, permite a los usuarios visualizar tanto datos almacenados como aquellos generados en tiempo real por los diversos dispositivos IoT.

El sistema de plantillas, diseñado para satisfacer las necesidades individuales de los usuarios, se respaldó en una investigación, que a través de casos de uso, confirmó la idoneidad de las gráficas seleccionadas para una variedad de sensores. Esta aproximación garantiza que la aplicación pueda abordar la mayoría de los requerimientos de los usuarios, independientemente de la diversidad de sensores involucrados.

En el ámbito del diseño visual, se adoptó Figma como herramienta principal para crear mockups de la aplicación web. Esta elección resultó fundamental, ya que proporcionó una guía coherente y ordenada para el desarrollo de la interfaz de usuario. A través de Figma, se establecieron directrices claras que ayudaron a transmitir eficazmente la finalidad de la aplicación y cómo los usuarios interactuarían con las gráficas que representan los datos de los sensores.

A pesar de que la arquitectura inicial carecía de un sistema de obtención de datos óptimo, se propuso una solución para mejorar este aspecto, permitiendo así el camino para futuras modificaciones del proyecto. Esta funcionalidad se implementó para la segunda versión de la arquitectura, lo que permite un desarrollo actualizado a los requerimientos de la arquitectura.

6. Trabajo futuro

Como trabajo futuro se plantearon algunas recomendaciones las cuales se desarrollaron a través del análisis profundo del funcionamiento y limitaciones del proyecto, a continuación se mencionan algunos de estos hallazgos a manera de recomendaciones:

El actual sistema de autenticación, basado en Spring Security, aunque funcional, plantea desafíos en cuanto a la complejidad de su arquitectura y flujo de procesos, lo que podría complicar futuras modificaciones y extensiones del sistema. Para solucionar esto se recomienda el uso de *Keycloak*, una herramienta que simplifica y automatiza la configuración del flujo de autenticación, permitiendo la gestión de accesos y permisos para usuarios en diferentes contextos. Cuenta con una interfaz gráfica personalizable para proceso de inicio de sesión de los usuarios.

Es importante destacar que *Keycloak* es altamente compatible con Spring Security, lo que simplifica su integración con la lógica de seguridad de los microservicios de la aplicación, aparte de esto, ofrece bibliotecas que facilitan la comunicación con aplicaciones Angular, lo que hace que su implementación sea aún más sencilla.

Aunque se implementó la obtención de los datos en tiempo real de los dispositivos mediante el tópico de “device-uuid”, esta fue implementada únicamente en el back de datos. Por lo tanto, se sugiere extender esta funcionalidad al Framework de *Gateway*, con el fin de realizar pruebas con dispositivos reales conectados a Smart Campus UIS.

El panel de control actualmente permite diseñar un panel de control con ilimitados bloques, algo que representa un problema que puede afectar negativamente el rendimiento y la organización de la interfaz. Por lo tanto, se plantea la necesidad de establecer un límite en la cantidad de bloques permitidos e implementar un sistema de pestañas o agrupadores, de esta manera, se mejoraría la experiencia del usuario al proporcionar una forma más ordenada y accesible de acceder a la información de los dispositivos.

Bibliografía

Arias, K., & Estupiñan, J. (2019). *Diseño del componente software backend orientado a una plataforma IoT diseñada para Smart Campus*.

AWS. (2023a). *¿Qué es una API?* Consultado el 20 de septiembre de 2023, desde https://aws.amazon.com/es/what-is/api/

AWS. (2023b). *Microservicios*. Consultado el 20 de septiembre de 2023, desde https://aws.amazon.com/es/microservices

de Málaga, U. (2020). *¿Qué es un Smart Campus?* Consultado el 20 de septiembre de 2023, desde https://www.uma.es/smart-campus/info/111661/i-plan-propio-de-smart-campus

Github. (2023). *Github Actions*. Consultado el 20 de septiembre de 2023, desde https://docs.github.com/en/actions/learn-github-actions/understanding-github-actions

Gracia, M. (2023). *¿Qué es IoT?* Consultado el 20 de septiembre de 2023, desde https://www2.deloitte.com/es/es/pages/technology/articles/IoT-internet-of-things.html

Gutiérrez, C. (2019). *Diseño de un framework software extensible para dispositivos tipo gateway integrados en plataformas IoT para Smart Campus*.

Pérez, S., & Claro, J. (2023). *Sistema de información basado en IoT para medición de la calidad del aire y seguimiento de rutas en sistemas de transporte público*.

Rojas, J. (2019). *Definición de una infraestructura cloud de alta disponibilidad en un entorno distribuido para el despliegue de una plataforma IoT*.

RTVE.es. (2010). *Niveles de decibelios (dB) en nuestro entorno*. Consultado el 19 de septiembre de 2023, desde https://www.rtve.es/noticias/20100328/niveles-decibelios-db-nuestro-entorno/322078.shtml

Tanzu, V. (2023). *Spring Security*. Consultado el 20 de septiembre de 2023, desde https://spring.io/guides/gs/securing-web/

Zettler, K. (2023). *¿Qué es un sistema distribuido?* Consultado el 20 de septiembre de 2023, desde https://www.atlassian.com/es/microservices/microservices-architecture/distributed-architecture