Diseño de una solución para la definición de escenarios de simulación de sensores en la plataforma Smart Campus UIS

Carlos Daniel Peñaloza Torres, Mariana Robayo Nieto

Trabajo de Grado para Optar al Título de Ingeniero de Sistemas

Director

Gabriel Rodrigo Pedraza Ferreira

PhD en Ciencias de la Computación

Codirector

Henry Andrés Jiménez Herrera

MsC en Ingeniería de Sistemas e Informática

Universidad Industrial de Santander

Facultad de Ingenierías Fisicomecánicas

Escuela de Ingeniería de Sistemas e Informática

Ingeniería de Sistemas

Bucaramanga

2025

Dedicatoria

A mi madre, Marinella Torres Sarmiento, cuya fortaleza, dedicación y amor incondicional han sido la luz y el cimiento de mi camino. Mi más profunda gratitud por cada sacrificio y enseñanza que han forjado mi vida.

A mi novia, Gabriela Galvis Gómez, por su amor, paciencia y apoyo constante. Gracias por estar a mi lado y por ser una parte importante en cada paso que doy. Cada logro alcanzado es también un reflejo de tu apoyo y dedicación.

Carlos Daniel Peñaloza Torres

A mi padre, por siempre creer en mí, apoyarme en cada paso y hacerme sentir capaz de todo, incluso cuando yo dudaba.

A mi madre, por ser el alma y la fuerza detrás de este logro. Su esfuerzo, sacrificio y apoyo incondicional han hecho posible este sueño.

A mis hermanas, por ser una fuente de inspiración y recordarme con cada uno de sus logros que el esfuerzo y la perseverancia abren cualquier camino.

A mi novio, por ser mi refugio en los momentos difíciles, por sostenerme cuando quise rendirme y por darme la fuerza para seguir adelante.

Esto es de ustedes y para ustedes. Los amo.

Mariana Robayo Nieto

Agradecimientos

Queremos expresar nuestro más sincero agradecimiento a todas las personas que hicieron posible la realización de este trabajo de grado.

A nuestros directores, PhD. Gabriel Rodrigo Pedraza Ferreira y MSc. Henry Andrés Jiménez Herrera, por su guía, conocimientos y dedicación durante el desarrollo de este proyecto.

A la Universidad Industrial de Santander por brindarnos los recursos y el espacio académico para crecer profesionalmente.

Y, por supuesto, a nuestras familias y amigos, por su apoyo incondicional, paciencia y motivación a lo largo de esta etapa.

Tabla de Contenido

Pág.

Lista de Tablas

Pág.

Lista de Figuras

Pág.

Lista de Apéndices

pág.

Glosario

AMQP: Advanced Message Queuing Protocol

API: Application Programming Interface

CLI: Command Line Interface

DSL: Domain Specific language

HTTP: Hypertext Transfer Protocol

IoT: Internet of Things

JSON: JavaScript Object Notation

MQTT: Message Queue Telemetry Transport

REST: Representational State Transfer

YAML: Yet Another Markup Language

Resumen

Título: Diseño de una solución para la definición de escenarios de simulación de sensores en la plataforma Smart Campus UIS[*]

Autor: Carlos Daniel Peñaloza Torres y Mariana Robayo Nieto[**]

Palabras Clave: IoT, Simulación, Sensores, Smart Campus

Descripción:

El desarrollo de aplicaciones IoT enfrenta el desafío de replicar condiciones operativas y ambientales de manera precisa durante las fases de implementación y validación. Tradicionalmente, las pruebas se realizan con prototipos físicos, lo cual limita el entorno de experimentación; alternativamente, se pueden desarrollar componentes de software que simulen las condiciones deseadas. Sin embargo, la primera opción no escala adecuadamente, ya que requiere grandes despliegues de infraestructura para emular entornos reales, mientras que la segunda implica desarrollos personalizados para cada nueva aplicación.

Este trabajo propone el diseño e implementación de un sistema de simulación que permita la creación de escenarios virtuales para sensores IoT, con el objetivo de facilitar el proceso de implementación y validación de aplicaciones en la plataforma Smart Campus UIS. La solución incluye un módulo de software capaz de emular diversas condiciones ambientales y operativas sin necesidad de hardware físico, reduciendo así los costos asociados. El sistema es flexible, ya que permite la conexión con plataformas que utilizan distintos protocolos, genera distribuciones de datos que simulan condiciones reales y adapta el formato de los datos a los requerimientos de la plataforma destino.

Además, el sistema es extensible, pues fue diseñado para permitir la incorporación de nuevos protocolos, condiciones de los dispositivos y distribuciones de datos. De esta forma, los desarrolladores podrán evaluar el comportamiento de sus aplicaciones en un entorno virtual controlado, simulando variables como temperatura, humedad, presencia de gases o detección de incendios, entre otras.

El sistema se integrará con la infraestructura del Smart Campus UIS, brindando a estudiantes e investigadores la posibilidad de realizar pruebas en escenarios virtuales que imitan condiciones reales del campus universitario. Esta herramienta contribuirá a optimizar la validación de tecnologías IoT, facilitando la toma de decisiones y promoviendo la investigación en un entorno accesible y flexible.

---

[*] Trabajo de Grado

[**] Facultad de Ingenierías Fisicomecánicas. Escuela de Ingeniería de Sistemas e Informática. Ingeniería de Sistemas. Director: Gabriel Rodrigo Pedraza Ferreira. PhD en Ciencias de la Computación. Codirector: Henry Andrés Jiménez Herrera. MsC en Ingeniería de Sistemas e Informática

Abstract

Title: Design of a Solution for Defining Sensor Simulation Scenarios on the Smart Campus UIS Platform.[*]

Author(s): Carlos Daniel Peñaloza Torres y Mariana Robayo Nieto[**]

Key Words: IoT, Simulation, Sensors, Smart Campus

Description:

Development of IoT applications faces the challenge of accurately replicating operational and environmental conditions during implementation and validation phases. Traditionally, testing is conducted using physical prototypes, which limits the experimentation environment. Alternatively, software components can be developed to simulate the desired conditions. However, the former approach does not scale well, as it requires extensive infrastructure deployments to emulate real-world environments, while the latter demands custom development for each new application.

This work proposes the design and implementation of a simulation system that enables the creation of virtual scenarios for IoT sensors, aiming to facilitate the implementation and validation processes of applications within the Smart Campus UIS platform. The solution includes a software module capable of emulating various environmental and operational conditions without the need for physical hardware, thereby reducing associated costs. The system is flexible, allowing integration with platforms that use different protocols, generating data distributions that simulate real-world conditions, and adapting data formats to meet the requirements of the target platform.

Furthermore, the system is extensible, as it was designed to support the incorporation of new protocols, device conditions, and data distributions. In this way, developers can evaluate the behavior of their applications in a controlled virtual environment, simulating variables such as temperature, humidity, gas presence, or fire detection, among others.

The system will be integrated with the Smart Campus UIS infrastructure, providing students and researchers with the ability to conduct tests in virtual scenarios that closely mimic real conditions on the university campus. This tool will contribute to optimizing the validation of IoT technologies, supporting decision-making, and promoting research in an accessible and flexible environment.

[*] Degree Work

[**] Faculty of Physical-Mechanical Engineering. School of Systems Engineering and Informatics. Systems Engineering. Director: Gabriel Rodrigo Pedraza Ferreira. PhD In Computer Science. Codirector: Henry Andrés Jiménez Herrera. MSc in Systems and Informatics Engineering.

Introducción

El término “Internet de las Cosas” (IoT) fue usado por primera vez por el británico Kevin Ashton en 1999 para describir un sistema en el cual los objetos del mundo real se podían conectar a internet por medio de sensores. Esta tecnología se ha convertido en una de las más importantes del siglo XXI y ha transformado por completo una amplia gama de sectores industriales incluidos el transporte, la manufactura, la salud y la electrónica de consumo. El IoT fusiona el mundo físico con el digital, permitiendo la recopilación y análisis de datos en tiempo real para optimizar procesos y mejorar la toma de decisiones. En el contexto del Smart Campus UIS, se ha identificado una anomalía significativa: la ausencia de un entorno adecuado para el desarrollo y validación de aplicaciones IoT. Actualmente, los desarrolladores dependen de prototipos físicos y entornos de prueba específicos, lo que no solo resulta costoso, sino que también limita la capacidad de simular una amplia variedad de condiciones operativas y ambientales. A esto se suma la falta de estandarización en las pruebas, lo que incrementa los costos y el tiempo de desarrollo. Esta situación se ve agravada por la inexistencia de asignaturas con enfoque específico en IoT y laboratorios dedicados exclusivamente a esta área en la universidad, lo que dificulta el avance de proyectos y la adquisición de habilidades técnicas en esta tecnología. Por lo tanto, surge la necesidad evidente de un componente de simulación que permita replicar diversas condiciones de manera eficiente y asequible, optimizando el proceso de desarrollo en el campus.

Para superar estas limitaciones, es fundamental contar con una herramienta de simulación que permita recrear escenarios realistas sin necesidad de depender de hardware especializado. En este sentido, plataformas como el Smart Campus UIS buscan integrar soluciones IoT en un entorno educativo y experimental, proporcionando una plataforma donde estudiantes, investigadores y

desarrolladores puedan probar sus aplicaciones. Para lograrlo, se necesita diseñar una solución avanzada que facilite la creación y gestión de escenarios de simulación personalizados, permitiendo evaluar el comportamiento de dispositivos IoT en condiciones variables sin recurrir a *hardware* costoso.

Este trabajo tiene como objetivo desarrollar un conjunto de funcionalidades que permitirán la simulación de sensores para el Smart Campus UIS, con el propósito de optimizar los procesos de prueba y validación. La propuesta implica el desarrollo de un *backend* encargado de la gestión de escenarios de simulación, así como el diseño de dos interfaces: una interfaz de línea de comandos y una interfaz web, con el fin de brindar a los usuarios herramientas accesibles para definir y personalizar sus simulaciones según las condiciones que desean evaluar. Además, se desarrolla un lenguaje de dominio específico declarativo que permite describir de forma sencilla escenarios de simulación y pruebas, eliminando la necesidad de programarlos en un lenguaje de programación tradicional y ofreciendo mayor flexibilidad a los desarrolladores.

El proceso de desarrollo se centró en crear una arquitectura modular y escalable, que pueda adaptarse a futuros cambios o expansiones del sistema. Además, se utilizó una estructura basada en contenedores y servicios desacoplados, lo que facilita la integración de nuevas funciones sin interrumpir el flujo de trabajo. Se adoptaron metodologías ágiles y se implementó integración continua para asegurar que cada nueva versión del sistema fuera probada de manera constante, manteniendo siempre su fiabilidad y rendimiento.

A lo largo del desarrollo, se realizaron numerosas pruebas funcionales, tanto en escenarios simulados como en entornos controlados dentro del Smart Campus UIS, para asegurarse de que el sistema pudiera replicar condiciones operativas reales. Estas pruebas también sirvieron para

identificar áreas de mejora y optimizar el rendimiento de la simulación, asegurando su eficacia como herramienta para la evaluación y validación de soluciones IoT.

Finalmente, se validó la aplicación del sistema dentro del Smart Campus UIS mostró resultados positivos, ya que no solo aceleró el proceso de validación de soluciones IoT, sino que también permitió probar una amplia gama de escenarios sin necesidad de contar con prototipos físicos costosos o limitados. Esto abre un abanico de posibilidades para futuras investigaciones y desarrollos, impulsando la innovación y mejorando la calidad de las soluciones tecnológicas que se implementan en el campus.

1 Objetivos

1.1 Objetivo General

Diseñar una funcionalidad para la definición y ejecución de escenarios de simulación de sensores con el fin de facilitar el desarrollo de aplicaciones IoT en la plataforma Smart Campus UIS a través de un componente software *backend* y sus interfaces.

1.2 Objetivos Específicos

Determinar los requerimientos funcionales y técnicos para la simulación de sensores en la plataforma *Smart Campus UIS*.

Diseñar la arquitectura de la solución de simulación basada en los requerimientos identificados.

Diseñar un lenguaje de dominio específico para la definición escenarios de simulación de sensores.

Desarrollar los componentes software que hacen parte de la solución, incluyendo, un *backend* de gestión, una interfaz de línea de comandos (CLI) y una interfaz web.

Validar la solución software implementada a través de un conjunto de pruebas funcionales de escenarios de simulación.

2 Marco de Referencia

El marco de referencia reúne los términos y tecnologías fundamentales que sustentan el diseño e implementación del sistema de simulación. Se definen conceptos claves, así como trabajos y desarrollos previos relacionados con la aplicación de IoT en entornos educativos. Esta es la base teórica que permite entender el enfoque general del proyecto.

2.1 Marco Conceptual

El marco conceptual define y explica los términos, conceptos y tecnologías que se utilizaran a lo largo del proyecto. Su propósito es establecer un lenguaje común y facilitar la comprensión del trabajo.

2.1.1 Internet de las Cosas

El concepto de Internet de las Cosas (IoT), fue introducido por Kevin Ashton en 1999 durante una presentación para la empresa Procter & G amble (P&G) para ilustrar el potencial de la tecnología RFID en la optimización de cadenas de suministro (Ashton, 2009).

Según la Internet Society (2015), Ashton lo utilizó para describir un sistema en el que objetos del mundo físico pueden conectarse a Internet mediante sensores, permitiendo la recopilación, el intercambio y el procesamiento de datos sin intervención humana.

2.1.2 Sensor

Kenny (2005) define un sensor como un dispositivo que convierte un fenómeno físico en una señal eléctrica, añadiendo que estos dispositivos representan la interfaz entre el mundo físico y los sistemas electrónicos. Estos dispositivos permiten la detección y medición de variables físicas o químicas, tales como temperatura, distancia, presión, humedad, luz, aceleración, desplazamiento, inclinación, entre otras.

En el ámbito del Internet de las Cosas (IoT), los sensores son muy importantes ya que permiten que los dispositivos recopilen información de su entorno, procesen datos en tiempo real y tomen decisiones automatizadas. Funcionan como el puente entre el mundo físico y el digital, facilitando la comunicación entre dispositivos y mejorando la eficiencia de distintos sistemas (InnovaciónTech, 2024).

2.1.3 Smart Campus

Según Polin, Yigitcanlar, Limb y Washington (2023), un Smart Campus es un entorno universitario que integra tecnologías avanzadas como el Internet de las Cosas (IoT), inteligencia artificial (IA) y sensores inteligentes para optimizar la gestión de recursos, mejorar la experiencia de estudiantes y personal, y fomentar la sostenibilidad. Se considera una extensión del concepto de ciudad inteligente, en la que la infraestructura tecnológica y la conectividad permiten una interacción eficiente entre personas, datos y servicios dentro del campus.

Por su parte, Coccoli (2014) amplía esta visión al señalar que un campus inteligente es aquel que no se limita únicamente a la implementación de tecnologías avanzadas, sino que también permite el intercambio de conocimientos entre empleados, profesores, estudiantes y todas las partes interesadas. Para lograrlo, deben priorizar la flexibilidad, la eficacia y el rendimiento, considerando las necesidades de la sociedad actual. Asimismo, destaca la importancia de establecer vínculos entre las universidades y la industria, con el fin de construir un ecosistema solido para convertirse en un campus inteligente.

2.1.4 Bróker de Mensajería

Según IBM un bróker de mensajería o agente de mensajes es un software que permite que las aplicaciones, los sistemas y los servicios se comuniquen entre sí e intercambien información. El intermediario de mensajes logra esto traduciendo mensajes entre protocolos de mensajería

formales. Esto permite que los servicios interdependientes se comuniquen directamente entre sí, incluso si están escritos en lenguajes diferentes o implementados en plataformas distintas.

Los intermediarios de mensajes pueden validar, almacenar, enrutar y entregar mensajes a los destinos adecuados. Actúan como intermediarios entre otras aplicaciones, permitiendo a los emisores enviar mensajes sin saber dónde están los receptores, si están activos o cuántos hay. Esto facilita la desvinculación de procesos y servicios dentro de los sistemas

2.1.5 Lenguaje de Dominio Específico

Un Lenguaje de Dominio Específico (DSL) es un lenguaje de programación diseñado con un alto nivel de abstracción para abordar problemas específicos dentro de un dominio particular. A diferencia de los lenguajes de propósito general como Java o C, los DSL están optimizados para facilitar la expresión de conceptos propios del dominio en cuestión, lo que permite mejorar la eficiencia y reducir la complejidad del desarrollo.

Los DSL pueden clasificarse en dos tipos: externos, cuando son independientes del lenguaje principal (como SQL), e internos, cuando se integran dentro de un lenguaje de propósito general. Su uso aporta beneficios como mayor claridad en la lógica de negocio, reducción de errores y facilidad de mantenimiento. Además, están alineados con la programación orientada a lenguaje, que promueve la creación y extensión de lenguajes adaptados a necesidades específicas (JetBrains, 2025).

2.1.6 Mustache

Mustache es un sistema de plantillas sin lógica (*logic-less templates*), lo que significa que solo se encarga de reemplazar variables y estructuras simples sin permitir lógica compleja dentro de la plantilla.

Se puede usar en integrar con diversos lenguajes de programación y es compatible con múltiples formatos como HTML, XML, archivos de configuración, entre otros. Los datos se pasan usualmente en formato JSON o YAML, y la plantilla se limita a mostrarlos según la estructura definida, sin evaluarlos ni transformarlos. Esto hace de Mustache una herramienta ligera, portable y predecible, ideal para contextos donde se requiere simplicidad y consistencia en la representación de datos.

2.1.7 MQTT (Message Queuing Telemetry Transport)

Es un protocolo de comunicación ampliamente utilizado en el contexto del Internet de las Cosas (IoT), diseñado específicamente para facilitar el intercambio de información entre maquinas (M2M). El mecanismo de transporte de mensajes de publicación/suscripción de MQTT es extremadamente ligero. Este protocolo es útil para establecer comunicaciones remotas cuando las velocidades de transferencia de datos son limitadas.

El protocolo MQTT admite la comunicación tanto síncrona como asíncrona, lo que propo rciona flexibilidad en la interacción entre dispositivos. Gracias a su simplicidad y bajo costo de implementación han llevado a su adopción en una variedad de aplicaciones, donde se requiere la gestión de grandes volúmenes de datos en entornos distribuidos.

2.1.8 AMQP (Advanced Message Queue Protocol)

El Protocolo de Cola de Mensajes Avanzado por sus siglas en inglés (AMQP) es un estándar de código abierto para IoT, diseñado para middleware orientado a mensajes. Utiliza comunicación asíncrona de publicación/suscripción con mensajería. Se trata de una función de almacenamiento y reenvío que garantiza la fiabilidad incluso tras interrupciones de red lo que constituye su principal ventaja.

AMQP es ampliamente utilizado en entornos inteligentes, procesos de automatización industrial y aplicaciones de atención médica, donde se requiere el intercambio de datos en tiempo real que provienen de sensores, comandos de control y notificaciones de eventos. Su capacidad para organizar los mensajes en colas estructuradas y la entrega garantizada lo convierte en una excelente opción para los ecosistemas de IoT.

2.1.9 Interfaz de Línea de Comandos

Una interfaz de línea de comandos por sus siglas en inglés (CLI) es un mecanismo de software que permite a los usuarios interactuar con el sistema operativo mediante el teclado. . A diferencia de una interfaz gráfica de usuario (GUI), la CLI no utiliza elementos visuales, sino que interpreta instrucciones escritas para realizar tareas como la configuración del sistema, la navegación entre archivos o la ejecución de programas en servidores u otros entornos informáticos.

2.2 Estado del Arte

En esta sección se recopilan estudios y desarrollos previos relacionados con la aplicación de tecnologías IoT en entornos educativos. La Tabla 1 presenta una selección de artículos sobre el desarrollo de campus inteligentes, resaltando enfoques, tecnologías y prácticas que sirven como base para la implementación del proyecto en la plataforma Smart Campus UIS.

Tabla 1.

Estado del Arte

Ref.	Título	Año	Revista	Contribución
[1]	**Development of Smart Campus Model**	**2021**	***International Conference on ICT for Smart Society (ICISS)***	Este artículo proporciona una comprensión más amplia del concepto de campus inteligente y cómo puede ser implementado en universidades. Al destacar cómo diversas instituciones han desarrollado modelos de campus inteligentes, como el iCampus del MIT o el Garuda Smart Campus Model en Indonesia, el artículo ofrece ejemplos y enfoques que pueden ser adaptados para la implementación en la plataforma Smart Campus UIS.
[2]	**People-Centric Smart Campus**	**2021**	***International Symposium on Computer Science and Intelligent Controls (ISCSIC)***	En nuestro proyecto, este enfoque contribuye a definir escenarios de simulación para el Smart Campus UIS, al asegurarse de que las soluciones tecnológicas desarrolladas no solo sean eficaces en términos operativos y respondan a las necesidades específicas de los usuarios. Así, se garantiza que las simulaciones reflejen de manera realista las condiciones y desafíos que enfrentan los usuarios, mejorando la validez y aplicabilidad de las soluciones IoT en el entorno universitario.
[3]	**Measurement of Campus Smartness: The Development of Smart Campus Model**	**2023**	***International Conference on ICT for Smart Society (ICISS)***	La metodología propuesta en el artículo, que incluye el uso de modelos como el Digital Twin para simular y mejorar las condiciones del campus, se alinea con el objetivo de desarrollar una funcionalidad para la simulación de escenarios. Esta perspectiva permite integrar y evaluar de manera efectiva las soluciones IoT en un entorno controlado y simulado, garantizando que estas soluciones sean robustas y eficaces antes de su implementación real en el campus.

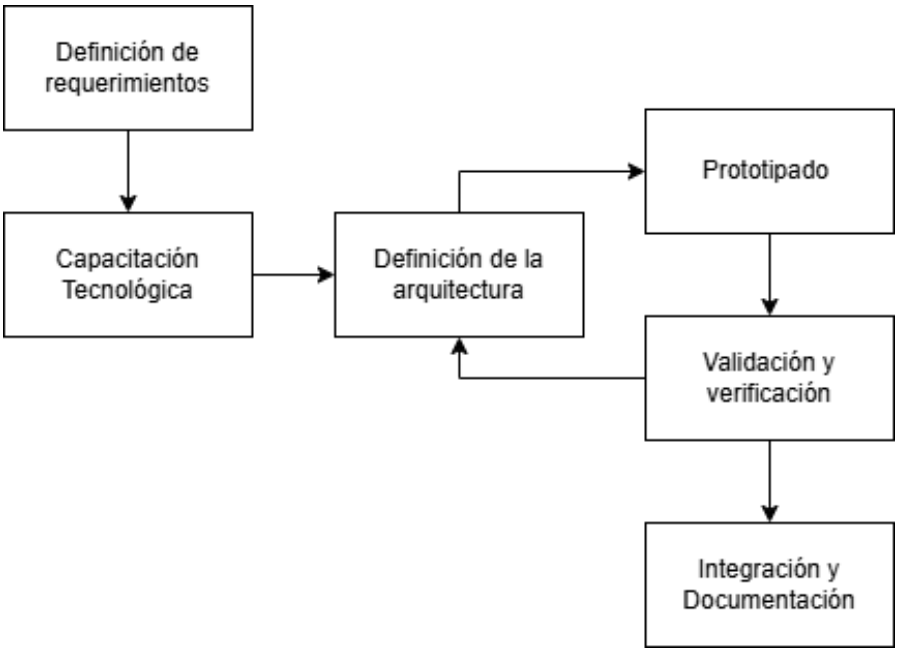
[4]	**A survey on internet of things based smart, digital green and intelligent campus**	**2019**	***International Conference on Internet of Things: Smart Innovation and Usages (IoT-SIU)***	Este artículo subraya la importancia de integrar diversas infraestructuras del campus, como sistemas de iluminación, seguridad y control climático, a través de IoT, lo que permite la optimización de recursos y mejora la experiencia educativa. Esta perspectiva refuerza la justificación de este proyecto al demostrar cómo la simulación de escenarios IoT puede facilitar el desarrollo y validación de soluciones innovadoras en el entorno del Smart Campus UIS, contribuyendo a un uso más eficiente de los recursos y una mayor sostenibilidad.

3 Metodología

Con el objetivo de garantizar una ejecución efectiva del proyecto y alcanzar los objetivos planteados, el trabajo ha sido estructurado en diversas fases que conforman nuestra metodología, basada en un enfoque de prototipo evolutivo. Esta división permite una organización eficiente de las tareas, los recursos y los tiempos, facilitando así un desarrollo sistemático y coherente del proceso general (véase Figura 1).

Figura 1

Esquema de metodología de trabajo.



3.1 Definición de Requerimientos

La primera fase de la metodología se basó en la identificación de los requerimientos necesarios para el desarrollo de la solución, asegurando así que todas las necesidades sean comprendidas y alineadas con los objetivos propuestos.

Las actividades realizadas en esta fase fueron:

- Elaborar un listado de especificaciones que detalle las funciones y características esenciales y deseables del Simulador de Sensores Smart Campus UIS, incluyendo sus clientes web y CLI.

3.2 Capacitación Tecnológica

La segunda fase consistió en la adquisición de conocimientos y habilidades necesarias para llevar a cabo el proyecto. Se centró en la investigación de las tecnologías y herramientas específicas que se utilizaron para diseñar, implementar y validar la solución. Se buscó familiarizar con diversos conceptos y practicas relacionadas con el Internet de las Cosas (IoT), la simulación de sensores y el desarrollo del *backend* y el *frontend*.

Las actividades realizadas en esta fase fueron:

- Investigación de fundamentos teóricos y estado del arte.
- Análisis y comprensión de las tecnologías y herramientas actuales utilizadas en la simulación de sensores.
- Identificación y selección de las tecnologías, *frameworks* y lenguajes de programación para el desarrollo de la solución de simulación en la plataforma Smart Campus UIS.
- Realización de cursos/tutoriales sobre las tecnologías, *frameworks* y lenguajes de programación seleccionados para el desarrollo de la solución.

3.3 Definición de la Arquitectura

Durante la tercera fase del proyecto, se buscó definir la arquitectura del sistema que brindó soporte a la solución identificando los componentes clave, sus interacciones y como estos se integraron para dar cumplimiento a los requerimientos del proyecto

Las actividades realizadas en esta fase fueron:

- Definición del alcance del proyecto y sus características.
- Análisis de requerimientos para determinar las necesidades arquitectónicas que detallan los componentes, tecnologías y patrones de diseño.
- Elaboración de un diagrama modular de la arquitectura que representa la arquitectura del sistema y su estructura interna.
- Integración y comunicación entre los diferentes módulos y su interacción con la plataforma Smart Campus UIS.

3.4 Prototipado

La cuarta fase del proyecto fue orientada a la implementación de la arquitectura definida. Posteriormente, se realizaron iteraciones sucesivas con el fin de mejorar el prototipo del sistema, hasta cumplir todos los requerimientos establecidos previamente.

Las actividades realizadas en esta fase fueron:

- Diseñar la interfaz de la plataforma web por medio de mockups para evaluar el cumplimiento de las características especificadas en los requerimientos.
- Definición y estructuración de la interfaz de la consola CLI, asegurando una experiencia de usuario intuitiva y funcional.
- Desarrollar un prototipo funcional que nos permitió hacer la simulación de sensores a través de la interacción con el usuario.

3.5 Validación y Verificación

La quinta fase se centró en la realización de pruebas del prototipo implementado, validando y verificando el funcionamiento y desempeño del prototipo en diversos escenarios.

Las actividades realizadas en estas fases fueron:

- Realización de pruebas funcionales para verificar la correcta interacción y desempeño del prototipo con la arquitectura y otros componentes del sistema.

3.6 Integración y Documentación

En la fase final del proyecto se realizó la integración del sistema con la plataforma Smart Campus UIS, asegurando su correcto funcionamiento. Además, se elaboró la documentación necesaria para garantizar un soporte adecuado y facilitar el uso del sistema.

Las actividades realizadas en esta fase fueron:

- Integración del sistema con la infraestructura existente en el Smart Campus UIS.
- Documentación técnica completa de la plataforma web, CLI, API y el DSL.

4 Requerimientos Funcionales y Técnicos

Este capítulo describe los requerimientos funcionales y técnicos del sistema propuesto. Los requerimientos funcionales definen las capacidades esenciales del simulador desde la perspectiva del usuario, mientras que los requerimientos técnicos establecen las condiciones necesarias para su correcto desempeño dentro del entorno de la plataforma Smart Campus UIS.

4.1 Requerimientos Funcionales

Los requerimientos funcionales definen las capacidades específicas que el sistema debe ofrecer para cumplir con los objetivos planteados. En esta sección se presentan los requerimientos esenciales relacionados con las interfaces Web, la línea de comandos (CLI) y la API REST, los cuales fueron organizados considerando la interacción de los distintos tipos de usuario con el sistema. Se incluyen funcionalidades como la carga y gestión de archivos de descripción de la simulación, la creación y control de simulaciones, el acceso a registros de actividad y la transmisión en tiempo real del estado de las simulaciones. Estas características fueron definidas a partir de un análisis del flujo operativo necesario para facilitar la simulación de sensores de forma flexible y accesible (véase Apéndice A).

4.2 Requerimientos No Funcionales

Los requerimientos no funcionales abarcan aquellas características del sistema que no están directamente relacionadas con funciones específicas, pero que afectan su calidad global. Estos incluyen criterios como rendimiento, portabilidad, mantenibilidad y usabilidad. En este proyecto, estos requerimientos son fundamentales para asegurar que la solución sea robusta, confiable y adaptable al crecimiento de la plataforma Smart Campus UIS, facilitando su adopción a largo plazo. (véase Apéndice B).

4.3 Requerimientos Técnicos

El sistema de simulación se desarrollará siguiendo una arquitectura moderna y escalable, garantizando una integración eficiente entre sus distintos componentes. A continuación, se detallarán los principales requerimientos técnicos:

- **Frontend**: La interfaz gráfica del sistema se desarrollará utilizando Angular, un *framework* basado en TypeScript, que permitirá la creación de aplicaciones web dinámicas y modulares.
- **Backend:** La lógica del sistema se implementará en Java, utilizando el *framework* Spring con Spring Boot, para la gestión de servicios REST, lo que asegurará un desarrollo estructurado y una comunicación eficiente entre los componentes.
- **Base de datos:** Se empleará MongoDB, una base de datos NoSQL, para el almacenamiento de información de las simulaciones. Su flexibilidad y capacidad de escalabilidad facilitarán el manejo eficiente de datos estructurados y semiestructurados.
- **Arquitectura:** El sistema adoptará los principios de Arquitectura Limpia, lo que permitirá una separación clara de responsabilidades, facilitando la mantenibilidad y escalabilidad del proyecto.
- **Actualización de datos:** Para proporcionar una experiencia fluida a los usuarios, la actualización de datos se realizará en tiempo real, asegurando que los cambios en las simulaciones se reflejen de manera inmediata.
- **CLI:** Se desarrollará una interfaz de línea de comandos (CLI) utilizando Picocli, un *framework* de Java que permitirá la creación de aplicaciones de consola interactivas y eficientes. La CLI permitirá a los usuarios gestionar simulaciones de manera sencilla, ejecutando comandos específicos para su configuración, ejecución y monitoreo.

- **DSL:** Se diseñará un Lenguaje de Dominio Específico (DSL) basado en YAML para definir las simulaciones. Este DSL permitirá la configuración detallada de protocolos, generadores de datos y estrategias de muestreo, facilitando la personalización de cada simulación sin necesidad de modificar el código fuente.
- **Contenedores:** Se utilizará Docker para empaquetar y ejecutar los distintos servicios del sistema en entornos aislados, garantizando portabilidad y consistencia en los despliegues.
- **Orquestación**: La gestión y coordinación de múltiples contenedores se realizará mediante Docker Compose, lo que permitirá una configuración centralizada y un despliegue eficiente de los servicios.
- **Alojamiento:** El sistema se podrá desplegar en servidores locales, asegurando compatibilidad con infraestructuras propias y facilitando su mantenimiento.
- **Gestión de configuración:** Las variables de entorno y configuraciones del sistema se administrarán a través del archivo docker-compose.yml, asegurando una configuración flexible y segura.

Una vez definidos los requerimientos funcionales y técnicos que guían el desarrollo del proyecto, se da paso al diseño de la solución. El siguiente capitulo aborda dos componentes clave: el diseño del Lenguaje de Dominio Específico (DSL), que permite representar de manera estructurada los escenarios de simulación, y la definición de la arquitectura, la cual sustenta la implementación técnica del sistema.

5 Diseño de la Solución

Este capítulo describe la solución propuesta a partir de los requerimientos definidos. Se presenta el diseño del Lenguaje de Dominio Específico que permite modelar los escenarios de simulación de forma estructurada, y la arquitectura del sistema, que define la organización de los componentes técnicos. Ambos elementos conforman la base para el desarrollo e implementación de la solución.

5.1 Diseño del Lenguaje de Dominio Especifico

Para definir de manera estructurada las simulaciones, se desarrolló un Lenguaje de Dominio Específico (DSL) basado en el formato YAML. Esta definición permite a los usuarios configurar de forma declarativa los distintos aspectos de una simulación, esto mediante un archivo de configuración compuesto por elementos clave que especifican los protocolos de comunicación, el tipo de muestreo y los generadores de datos (véase Figura 2).

Los protocolos representan el mecanismo mediante el cual los datos simulados son transmitidos hacia plataformas o sistemas externos como el Smart Campus UIS. En esta solución se incluyen protocolos ampliamente utilizados en entornos IoT como MQTT y AMQP. El DSL permite seleccionar y parametrizar cualquiera de estos protocolos, definiendo aspectos como el servidor de destino, tópicos de publicación o credenciales de acceso.

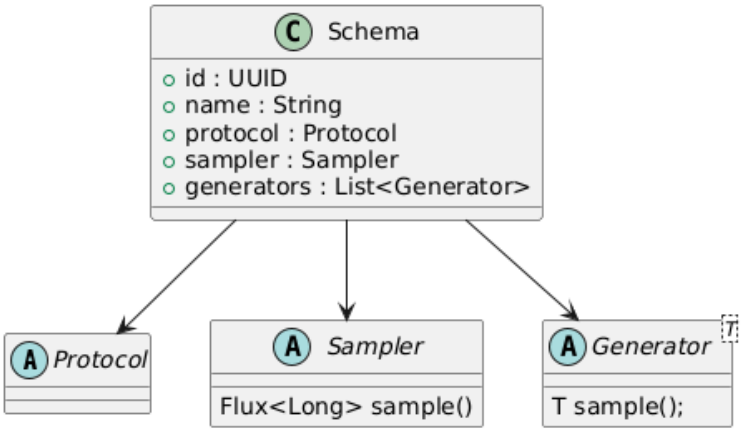
El componente de muestreo define la frecuencia o el patrón temporal con el cual los datos son generados y enviados. Es decir, determina “cuándo” ocurre cada emisión de datos dentro del escenario simulado. Este muestreo puede ser periódico, aleatorio dentro de un rango de tiempo, o basado en distribuciones. La posibilidad de personalizar el muestreo permite modelar con mayor

precisión distintas condiciones de operación, como sensores que envían datos continuamente o aquellos que solo se activan bajo ciertas condiciones.

Finalmente, los generadores son los componentes encargados de producir los datos que representan las lecturas de los sensores. Cada generador puede configurarse para emitir distintos tipos de valores según el tipo de sensor simulado. Se han definido generadores para datos booleanos, numéricos y cadenas de caracteres. Además, estos valores pueden seguir diferentes distribuciones estadísticas, como distribuciones uniformes, normales o personalizadas, lo que permite simular comportamientos realistas o condiciones anómalas. Cada generador incluye parámetros ajustables como rango de valores, unidades, precisión, o probabilidad de ocurrencia.

Figura 2

Diagrama del DSL.

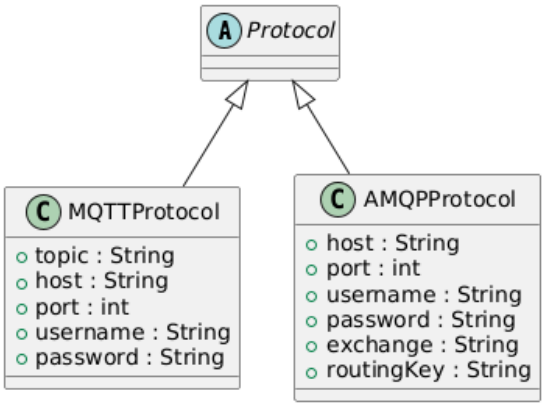


5.1.1 Protocolo

La sección "*protocol*" en el DSL permite definir de manera estructurada los parámetros de comunicación del sistema. Esta parte del lenguaje permitió configurar diversos protocolos, lo que facilitó la integración y el intercambio de información entre los componentes en distintos escenarios de simulación (véase Figura 3).

Figura 3

Diagrama de clases del componente Protocol.



Para la comunicación del sistema con plataformas externas, se seleccionó el protocolo MQTT debido a su uso extendido en aplicaciones IoT, donde destaca por su eficiencia, bajo consumo de ancho de banda y arquitectura basada en publicación/suscripción. Esta elección permite al sistema integrarse con plataformas en tiempo real. En la Tabla 2 se presentan los parámetros requeridos para configurar correctamente este protocolo dentro del archivo de descripción de la simulación, asegurando compatibilidad con *brokers* MQTT estándar como Mosquitto o EMQX.

Tabla 2

Protocolo MQTT.

Parámetro	**Descripción**
type	Tipo de protocolo (en este caso: “mqtt”).
host	Dirección del servidor.
port	Puerto del servidor.
topic	*Topic* por el cual se enviarán los mensajes.

clientId	Identificador del cliente.
username	Nombre de usuario para autenticación.
password	Contraseña para autenticación.

El protocolo AMQP se incluyó en el sistema por su capacidad de enrutamiento avanzado de mensajes, gestión robusta de colas y soporte para arquitecturas empresariales. Este protocolo es especialmente útil cuando se necesita un mayor control en la entrega de mensajes o se integra con middleware como RabbitMQ. En la Tabla 3 se muestran los parámetros necesarios para su configuración.

Tabla 3

Protocolo AMQP.

Parámetro	**Descripción**
type	Tipo de protocolo (en este caso: “amqp”)
host	Dirección del servidor.
port	Puerto del servidor.
username	Nombre de usuario para autenticación.
password	Contraseña para autenticación.
exchange	Nombre del *exchange*.
routingKey	Clave de enrutamiento para los mensajes.

5.1.2 Muestreador

Se definió la sección de muestreadores en el DSL para estructurar el proceso de muestreo de datos en las simulaciones. Este componente permite determinar el ritmo y la forma en que se generan las muestras, adaptándose a los requisitos específicos de cada escenario. Con este enfoque,

el sistema puede controlar de manera precisa el intervalo de tiempo entre muestras de las simulaciones. El siguiente diagrama muestra la estructura de clases correspondiente a los muestreadores, ilustrando su estructura y relaciones (véase Figura 4). En la Tabla 4 se describen los tipos de muestreadores junto con sus parámetros esenciales, los cuales fueron diseñados para brindar una mayor diversidad de estrategias para la adquisición de datos simulados, adaptándose a los requisitos específicos de cada posible escenario de simulación.

Figura 4

Diagrama de clases del componente Sampler

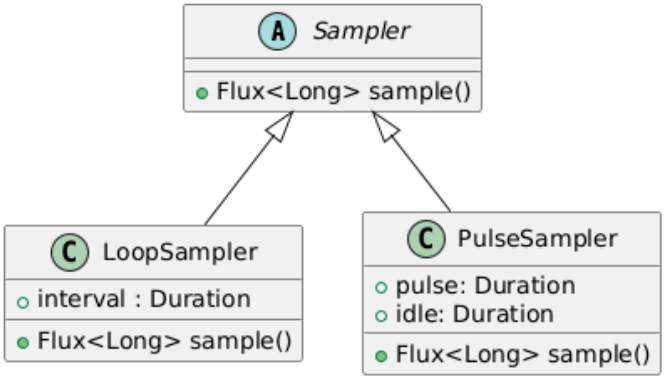


Tabla 4

Tipos de muestreadores.

Tipo	Parámetros	Descripción
step	duration, interval	Define un paso con una duración y un intervalo.
burst	delay, size	Envía un grupo de mensajes en ráfaga.
delay	delay	Introduce un retardo antes de continuar.
count	delay, count	Envía un número específico de mensajes.
random	min, max	Introduce un retardo aleatorio entre mensajes.

loop	interval	Repite el paso anterior un número indefinido de veces.
pulse	pulse, idle	Envía mensajes en pulsos.
traffic-spike	normal, spike, duration	Simula un pico de tráfico.
window	active, inactive, interval	Define una ventana de actividad e inactividad.

5.1.3 Generadores

Se estableció la sección de generadores en el DSL para la generación de datos en la simulación. Se definieron distintos tipos de generadores, tales como aquellos para producir valores booleanos, numéricos e incluso cadenas de caracteres o con distribuciones estadísticas específicas. Este componente representa el núcleo del sistema, ya que es responsable de producir los datos que emulan el comportamiento de los sensores simulados.

Con el fin de cubrir una amplia gama de posibles escenarios, se diseñaron generadores con distintos niveles de complejidad. Cada uno de estos tipos fue seleccionado por su representatividad en contextos reales y por su capacidad para combinarse con otros elementos del lenguaje, como los muestreadores, con el propósito de construir simulaciones más fieles a las condiciones operativas de dispositivos IoT en entornos diversos.

En particular, la incorporación de generadores basados en distribuciones estadísticas responde a la necesidad de reproducir patrones de comportamiento propios de variables físicas y fenómenos naturales. Por ejemplo, la variación de temperatura en espacios interiores puede aproximarse mediante una distribución normal, mientras que eventos discretos como la activación de sensores de presencia pueden modelarse con distribuciones de tipo Bernoulli o Poisson. Esta

capacidad de representar fenómenos mediante distribuciones matemáticas permite simular tanto comportamientos normales como situaciones atípicas o anómalas.

Además, esta estrategia proporciona una ventaja significativa en términos de flexibilidad y control experimental, al permitir parametrizar características como media, desviación estándar, probabilidad de ocurrencia, entre otras. De esta forma, los datos generados no solo son verosímiles, sino también reproducibles y ajustables según el caso de estudio, lo cual resulta especialmente útil en contextos de validación, pruebas automatizadas y entrenamiento de algoritmos de análisis de datos.

El siguiente diagrama muestra la estructura de clases correspondiente a la implementación de estos generadores, ilustrando su estructura y relaciones (véase la Figura 5). La Tabla 5, por su parte, detalla la estructura y los parámetros de cada tipo de generador, permitiendo su integración personalizada en distintos escenarios de simulación, conforme a las necesidades específicas de cada experimento.

Figura 5

Diagrama de clases del componente Generator.

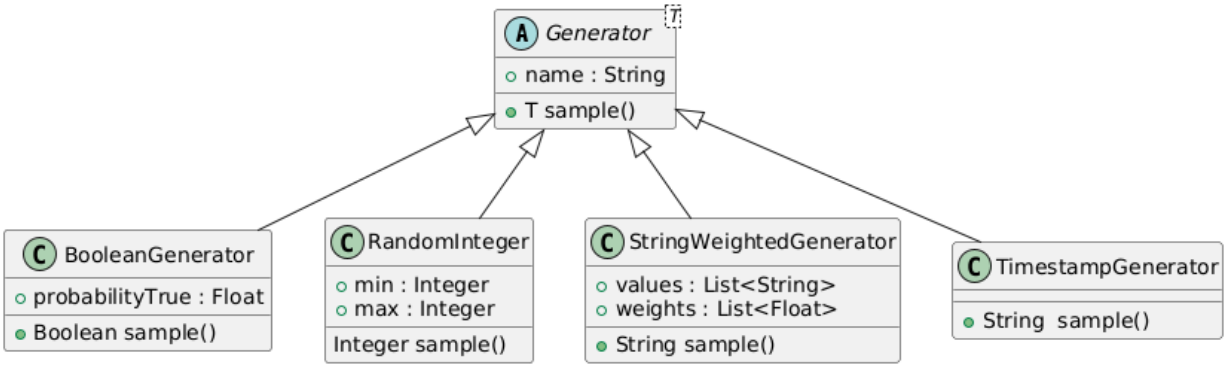


Tabla 5

Tipos de Generadores.

Tipo	Parámetros	Descripción
timestamp	Name	Genera una marca de tiempo.
boolean	name, probability	Genera un valor booleano.
random_integer	name, min, max	Genera un número entero aleatorio.
random_double	name, min, max, decimals	Genera un número decimal aleatorio.
continuous_exponential	name, lambda, decimals	Genera valores de una distribución exponencial.
continuous_log_normal	name, mean, stddev, decimals	Genera valores de una distribución log-normal.
continuous_normal	name, mean, stddev, decimals	Genera valores de una distribución normal.
continuous_triangular	name, min, max, mode, decimals	Genera valores de una distribución triangular.
continuous_uniform	name, min, max, decimals	Genera valores de una distribución uniforme continua.
discrete_bernoulli	name, probability	Genera valores de una distribución de Bernoulli.
discrete_binomial	name, trials, probability	Genera valores de una distribución binomial.
discrete_geometric	name, probability	Genera valores de una distribución geométrica.
discrete_poisson	name, lambda	Genera valores de una distribución de Poisson.
discrete_uniform	name, min, max	Genera valores de una distribución uniforme discreta.
string	name, sampling, values, weights	Genera cadenas de texto.

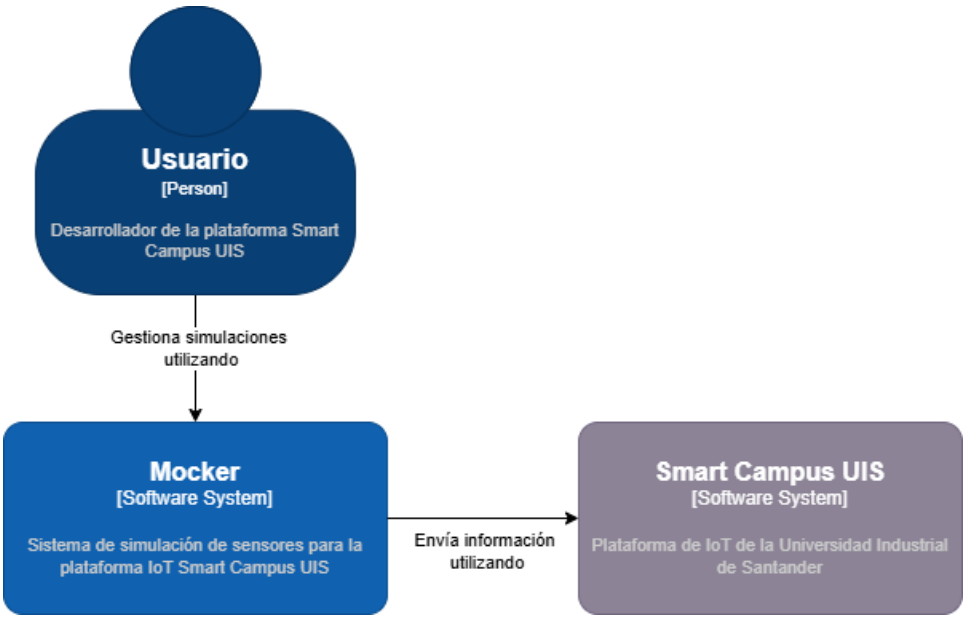
5.2 Definición de la Arquitectura

Para entender la arquitectura general del sistema, se diseñó un diagrama de contexto en el que se identifican los principales actores y su interacción con el sistema. En este nivel, se representa al usuario como la entidad que interactúa con la solución, ya sea a través de una interfaz web o mediante una interfaz de línea de comandos (CLI). Esta interfaz se comunica con un *backend* que gestiona la lógica de la simulación y persiste la información en una base de datos.

Además, se muestra cómo el sistema se conecta con componentes externos como el Smart Campus UIS mediante protocolos de mensajería como MQTT o AMQP para enviar datos generados por la simulación. El siguiente diagrama ilustra esta arquitectura general, destacando las relaciones entre los actores y los principales componentes del sistema (véase Figura 6).

Figura 6

Diagrama de contexto del sistema.

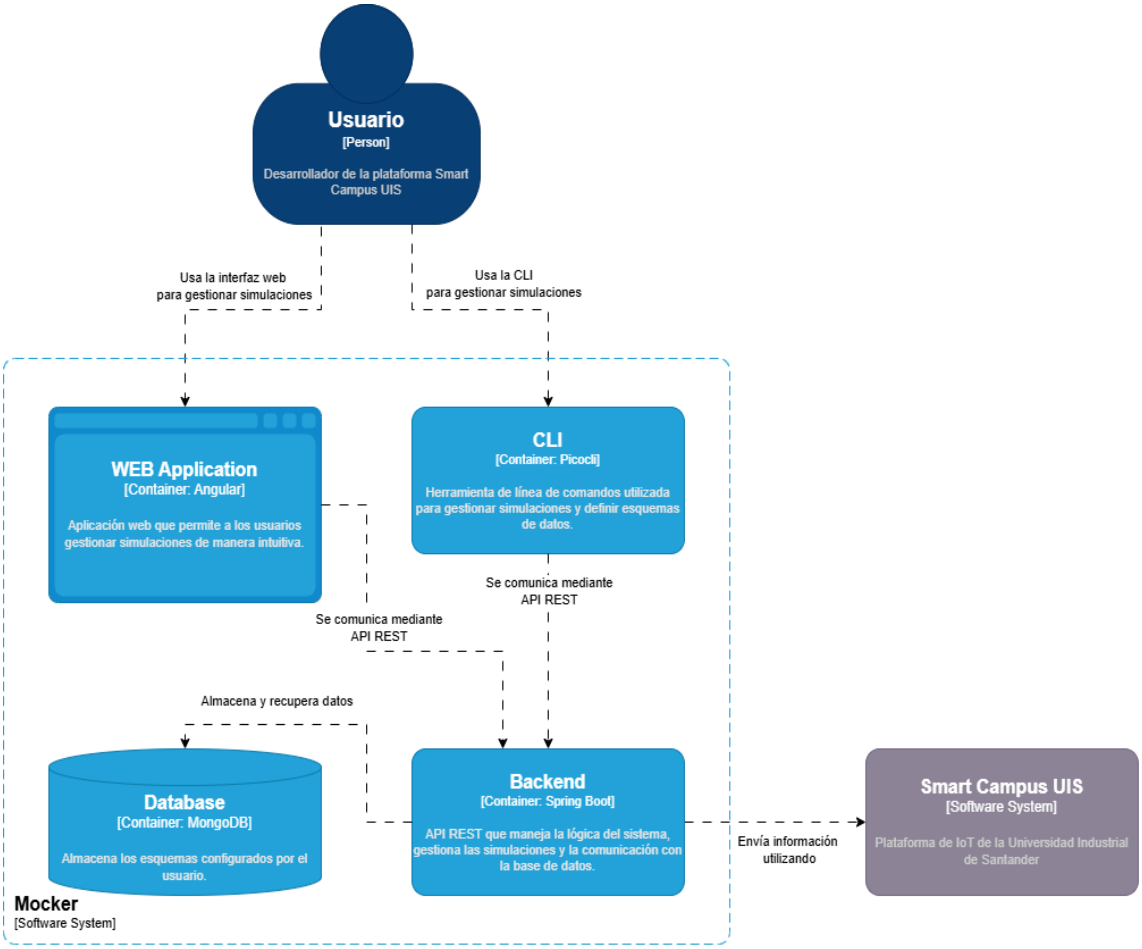


Para profundizar en la estructura del sistema, se detalla los principales contenedores de la aplicación. En este diagrama (véase Figura 7), se identifica el *frontend*, que puede ser una aplicación web o una interfaz CLI, encargada de la interacción con el usuario. También se muestra el *backend*, el cual es responsable de administrar las simulaciones, procesar eventos y gestionar la comunicación con sistemas externos. La base de datos se encarga de almacenar los archivos de descripción de la simulación definidos por los usuarios. Finalmente, se destaca la integración con

el Smart Campus UIS, que recibe los datos generados en tiempo real mediante los protocolos anteriormente definidos.

Figura 7

Diagrama de contenedores del sistema.

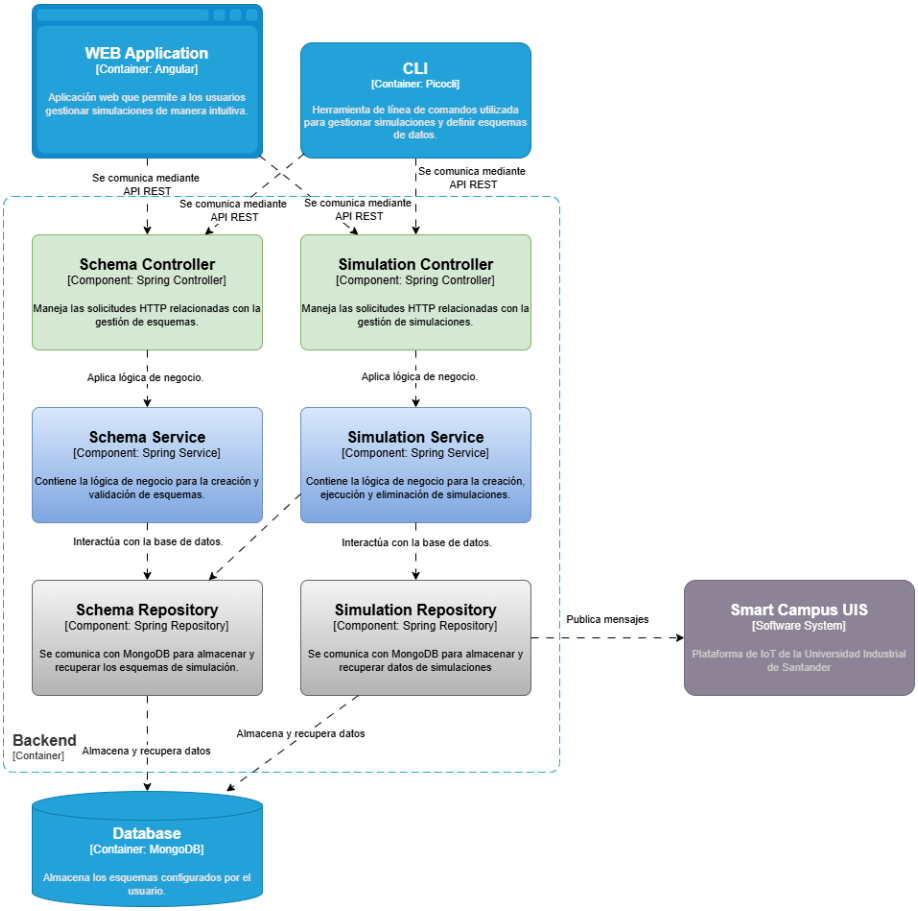


La estructura interna del *backend*, enfocada en la organización del código y la interacción entre sus componentes principales puede visualizarse en el siguiente diagrama (véase Figura 8). En este nivel, se representan controladores, servicios, repositorios y se describe la manera en que estos se relacionan. El controlador recibe las solicitudes de la aplicación (ya sea desde la interfaz web o la CLI) y las redirige al servicio, que implementa la lógica de negocio. A su vez, el servicio

se comunica con los repositorios, los cuales gestionan el acceso a la base de datos para almacenar o recuperar la información de las simulaciones y archivos de descripción.

Figura 8

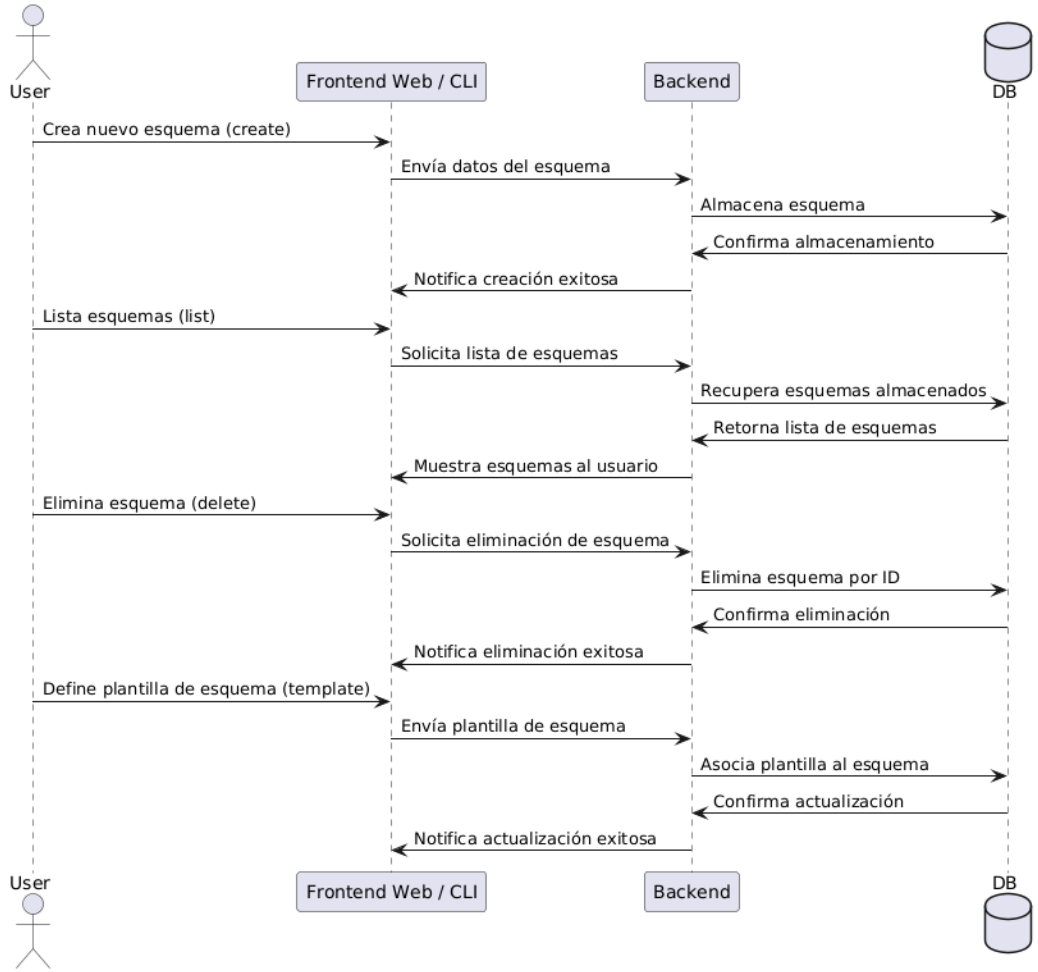
Diagrama de componentes del sistema.



Para gestionar los archivos de descripción de la simulación, se implementó un mecanismo de CRUD (*Create*, *Read*, *Update*, *Delete*). A continuación, se presenta el diagrama de secuencia correspondiente, en el que se detalla la interacción entre el usuario, la interfaz (CLI o Web), el *backend* y la base de datos (véase Figura 9).

Figura 9

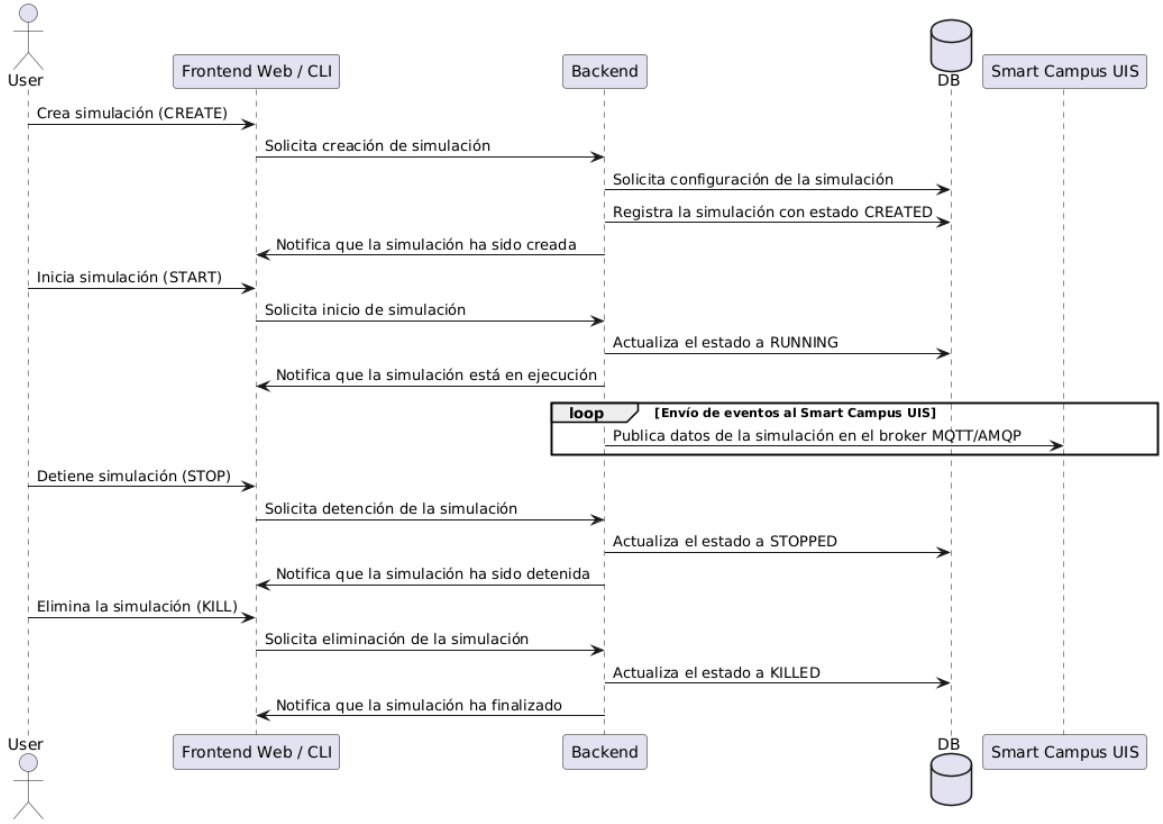
Diagrama de Secuencia - CRUD de esquemas.



El diagrama de secuencia (véase Figura 10), muestra cómo el usuario inicia una simulación a través de la interfaz (CLI o Web), lo que genera una solicitud al *backend* para recuperar el archivo de descripción de la simulación de la base de datos y poder crear y ejecutar la simulación. Una vez que la simulación está en ejecución, se inicia un ciclo en el cual se generan y envían datos continuamente al Smart Campus UIS a través de los protocolos definidos. En caso de que el usuario detenga (STOP) o elimine (KILL) la simulación, el *backend* interrumpe el proceso de envío de datos y ajusta el estado de la simulación.

Figura 10

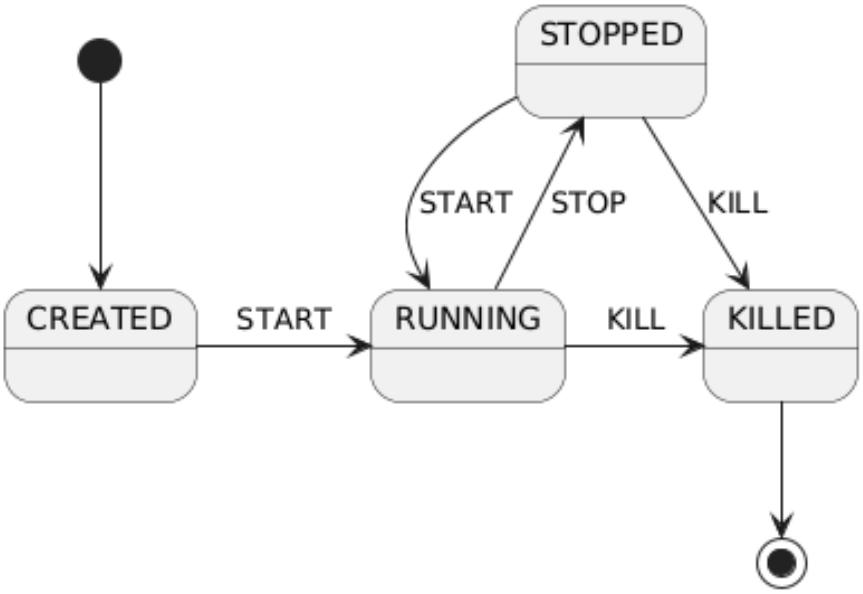
Diagrama de Secuencia - Flujo de Simulación.



Se implementó una máquina de estados para definir claramente las etapas del ciclo de vida de la simulación, abarcando todos los estados posibles. Como se muestra en el diagrama (véase Figura 11), la simulación inicia en el estado CREATED, en el cual se encuentra preparada pero aún no en ejecución. Al recibir un evento START, cambia al estado a RUNNING, iniciando el envío de datos al Smart Campus UIS. Si se recibe un evento STOP, la simulación pasa a STOPPED, pausando el envío de datos sin eliminar la simulación. Finalmente, si se ejecuta un evento KILL, la simulación finaliza de manera permanente, entrando en el estado KILLED, para posteriormente liberar los recursos del sistema y concluir el proceso.

Figura 11

Diagrama de Estados - Ciclo de Vida de la Simulación.



Concluido el proceso de diseño, en el cual se establecieron los lineamientos estructurales y funcionales de la solución propuesta, se da paso a su correspondiente implementación. El siguiente capítulo expone de manera detallada el desarrollo de la solución, abordando los aspectos técnicos y metodológicos empleados para llevar a cabo la construcción del sistema. Esta fase representa un paso fundamental, ya que permite materializar los conceptos definidos previamente y establecer las bases necesarias para su validación mediante pruebas y análisis posteriores.

6 Implementación de la Solución

Para la implementación de la solución, se desarrollaron tres componentes principales: el *backend*, la CLI y la interfaz web, cada uno diseñado para cumplir un rol específico dentro de la arquitectura. El *backend* se encargó de procesar las simulaciones, gestionar la configuración definida en el DSL y coordinar la comunicación con la base de datos. La CLI permitió la ejecución y administración de simulaciones desde un entorno de línea de comandos. Finalmente, la interfaz web facilitó la interacción con el sistema a través de una plataforma visual intuitiva, permitiendo la configuración y monitoreo de las simulaciones de manera accesible y eficiente. Las tecnologías utilizadas para esta implementación fueron previamente estudiadas en la fase de capacitación tecnológica (ver Apéndice C).

6.1 Backend

La implementación del *backend* se realizó utilizando el *framework* Spring Boot, debido a su robustez, modularidad y compatibilidad con arquitecturas modernas basadas en servicios. Esta capa se encarga de la lógica de negocio central del sistema, incluyendo la gestión de archivos de descripción, el control de simulaciones y la comunicación con la base de datos MongoDB. La integración con MongoDB, una base de datos NoSQL orientada a documentos, se facilitó mediante Spring Data MongoDB, lo que permitió un manejo eficiente y flexible de la información utilizada por las simulaciones.

6.1.1 Arquitectura

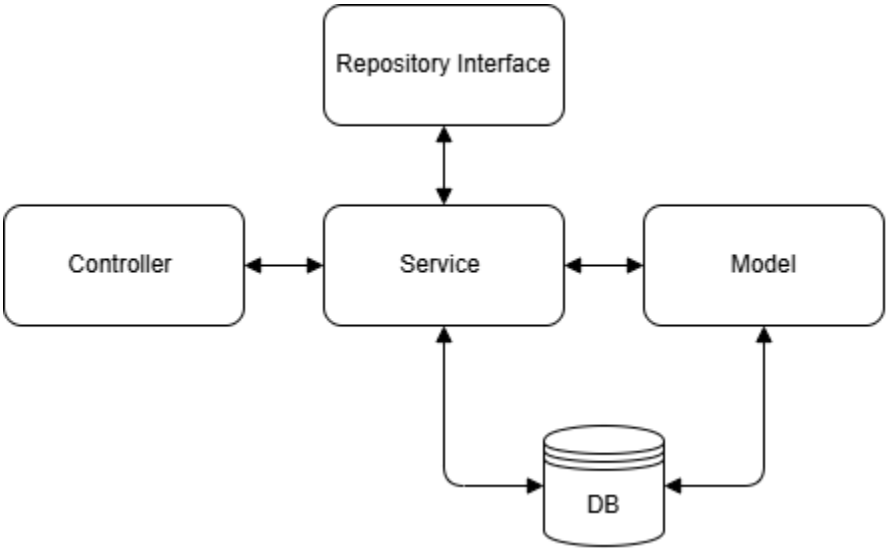
Se adoptó una arquitectura por capas, separando claramente las responsabilidades en cuatro niveles:

- **Controller:** expone una API REST siguiendo principios estándar para facilitar la interoperabilidad con otros sistemas.
- **Service:** implementa la lógica de simulación, validación de datos y orquestación de componentes.
- **Repository:** se conecta con MongoDB usando Spring Data para acceder y almacenar archivos de descripción de simulaciones.
- **Modelo:** define las entidades principales, como Simulation y Schema.

Esta organización se representa en la ilustración correspondiente a la estructura de capas adoptada (véase Figura 12).

Figura 12

Flujo de arquitectura de aplicaciones de Spring Boot.



Esta organización facilita el mantenimiento, la reutilización de código y la extensión del sistema (véase Apéndice D para más detalles sobre la organización modular y la arquitectura del *backend*).

6.1.2 API REST

La API se diseñó siguiendo los principios REST, utilizando los verbos HTTP de forma semántica (GET, POST, DELETE, etc.) y rutas bien definidas para cada recurso. Esto garantiza una fácil integración con clientes externos y mantiene la interoperabilidad con estándares modernos (véase Apéndice E).

6.1.3 Gestión de la Persistencia

La base de datos utilizada es MongoDB, elegida por su modelo flexible orientado a documentos, ideal para estructuras de datos heterogéneas como las de los archivos de descripción de simulaciones.

6.1.4 Validación y Manejo de Errores

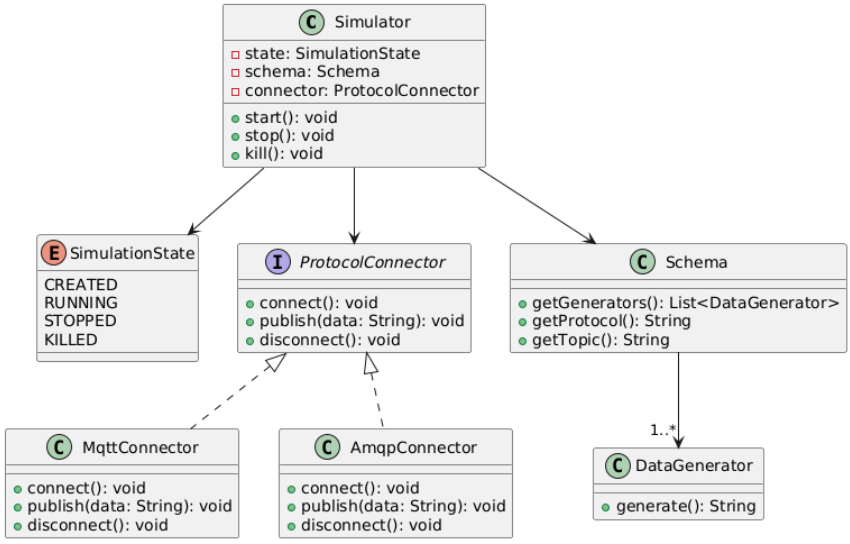
Se integró la validación automática de datos mediante anotaciones de javax.validation, como @NotNull y @Size, lo que garantiza la calidad de los datos recibidos desde los clientes en los archivos de descripción. El manejo de errores se centraliza en una clase @ControllerAdvice, que captura excepciones específicas y devuelve respuestas estructuradas con códigos de estado apropiados (400, 404, 500, etc.).

6.1.5 Simulador

El Simulador es un componente clave en el sistema, responsable de gestionar el ciclo de vida de las simulaciones, incluyendo su inicio, detención y la conexión con los protocolos necesarios. Los conectores MqttConnector y AmqpConnector se encargan de la comunicación con los protocolos MQTT y AMQP, respectivamente. Además, el Schema proporciona la configuración y los generadores necesarios para crear y procesar los datos de la simulación. . La relación entre estos elementos se muestra en el siguiente esquema (véase Figura 13).

Figura 13

Componentes del simulador.



El componente Simulador utiliza un enfoque basado en una máquina de estados finitos para gestionar de forma controlada el ciclo de vida de una simulación. Este enfoque permite representar los distintos estados por los que transita una simulación desde su creación hasta su terminación, así como las transiciones entre ellos mediante eventos definidos. La lógica de este comportamiento se presenta en el esquema correspondiente (véase Figura 11). Por su parte la Tabla 6 describe cada uno de los estados posibles dentro del ciclo de vida, mientras que la Tabla 7 detalla las transiciones permitidas entre estados y los eventos que las desencadenan.

Tabla 6

Estados del Simulador.

Estado	Descripción
created	Estado inicial tras crear una simulación. Aún no se ha iniciado.
running	Simulación activa, generando y transmitiendo datos.

stopped	Simulación detenida de forma controlada. Puede reiniciarse.
killed	Simulación finalizada abruptamente por error o intervención forzada.

Tabla 7

Eventos y Transiciones del simulador.

Evento	Transición	Acción esperada
start	created → running	Inicia la generación de datos.
stop	running → stopped	Detiene temporalmente la simulación.
kill	created → killed	Finaliza la simulación sin posibilidad de recuperación.
kill	running → killed	Finaliza la simulación sin posibilidad de recuperación.

6.1.6 Formato del Mensaje

El *backend* del sistema incorpora el motor de BACs Mustache para facilitar la generación dinámica de mensajes de simulación. Esta funcionalidad es esencial para permitir que los datos generados por las simulaciones sean formateados y enviados de manera estructurada, siguiendo configuraciones personalizadas definidas por el usuario.

Cada simulación puede estar asociada a una plantilla Mustache que define el formato del mensaje a enviar. Estas plantillas utilizan variables que se sustituyen en tiempo de ejecución con los valores generados por los sensores simulados. A continuación, se muestra un ejemplo de plantilla Mustache utilizada para estructurar el mensaje (véase Figura 14), así como el resultado generado tras el reemplazo dinámico de las variables (véase Figura 15).

Figura 14

Ejemplo de plantilla Mustache.

```
{
  "temperature": {{temperature}},
  "timestamp": "{{timestamp}}"
}
```

Figura 15

Ejemplo de mensaje compilado.

```
{
  "temperature": 24.7,
  "timestamp": "2025-04-05T10:32:15Z"
}
```

6.2 Interfaz de Línea de Comandos

La Interfaz de Línea de Comandos (CLI) fue diseñada como un componente clave para facilitar la interacción técnica con el sistema de simulación de sensores, permitiendo a desarrolladores y usuarios avanzados ejecutar operaciones directamente desde la consola. La implementación se realizó utilizando la librería Picocli, combinada con el Spring Framework, lo que permitió una arquitectura robusta, extensible y alineada con los principios del desarrollo modular.

Gracias a Picocli, la CLI adopta una estructura basada en comandos y subcomandos jerárquicos, lo que facilita tanto su uso como su mantenimiento. Por otro lado, la integración con Spring permitió aplicar principios como la inyección de dependencias, la gestión del ciclo de vida de los *beans*, y el acceso directo a servicios del *backend*, manteniendo la consistencia con el resto del sistema.

6.2.1 Comando mocker

La CLI se estructura en torno a un comando principal (mocker) que ofrece dos grandes grupos de funcionalidades: simulation y schema. Cada uno de estos comandos cuenta con subcomandos específicos que permiten realizar acciones particulares sobre simulaciones y esquemas respectivamente (véase Figura 16).

Figura 16

Salida del comando principal de la CLI (mocker -h).

```
Usage: mocker [-hV] [COMMAND]
Mocker CLI tool
  -h, --help      Show this help message and exit.
  -V, --version   Print version information and exit.
Commands:
  simulation, sim  Manage simulations.
  schema, s        Manage schemas.
```

6.2.2 Comando schema

El comando schema permite la gestión de archivos de descripción de la simulación, los cuales representan las configuraciones estructuradas que definen cómo debe comportarse una simulación. Los esquemas incluyen elementos como generadores, protocolos de comunicación y muestreadores (véase Figura 17). Los subcomandos disponibles para operar sobre los esquemas y sus respectivas descripciones se detallan en la Tabla 8.

Figura 17

Ayuda del comando "schema".

```
Usage: mocker schema [-hV] [COMMAND]
Manage schemas.
  -h, --help      Show this help message and exit.
  -V, --version   Print version information and exit.
Commands:
  create, c    Create a new schema.
  list, ls     List all schemas.
  delete, d    Delete a schema.
  template, t Set the schema template.
```

Tabla 8

Listado de subcomandos del comando "schema".

Subcomando	Descripción	Ejemplo de uso
create	Crea un nuevo esquema de simulación a partir de un archivo.	mocker schema create --file <file.yaml>
list	Muestra todos los esquemas almacenados en el sistema.	mocker schema list
template	Define una plantilla de esquema para facilitar la creación posterior.	mocker schema template --file <file.txt>
delete	Elimina un esquema identificado por su UUID.	mocker schema delete --id <UUID>

6.2.3 Comando simulation

El comando simulation permite gestionar de forma integral el ciclo de vida de las simulaciones dentro del sistema (véase la Figura 18). A través de este comando, los usuarios pueden crear nuevas simulaciones, iniciar su ejecución, detenerlas, finalizarlas forzada o simplemente listarlas. Esta funcionalidad resulta fundamental para interactuar con el motor de simulación desde la línea de comandos, facilitando su uso en entornos automatizados o sin interfaz gráfica. Los subcomandos disponibles y sus usos se detallan en la Tabla 9.

Figura 18

Ayuda del comando "simulation".

```
Usage: mocker simulation [-hV] [COMMAND]
Manage simulations.
  -h, --help      Show this help message and exit.
  -V, --version   Print version information and exit.
Commands:
  create, c   Create a new simulation.
  list, ls    List all simulations.
  start, run  Start a simulation.
  stop        Stop a simulation.
  kill, k     Kill a simulation.
```

Tabla 9

Listado de subcomandos del comando "simulation".

Subcomando	Descripción	Ejemplo de uso
create	Crea una nueva simulación a partir de un esquema.	mocker simulation create --id <UUID>
start	Inicia una simulación previamente creada.	mocker simulation start --id <UUID>
stop	Detiene una simulación en ejecución.	mocker simulation stop --id <UUID>
kill	Finaliza forzosamente una simulación.	mocker simulation kill --id <UUID>
list	Muestra todas las simulaciones registradas.	mocker simulation list

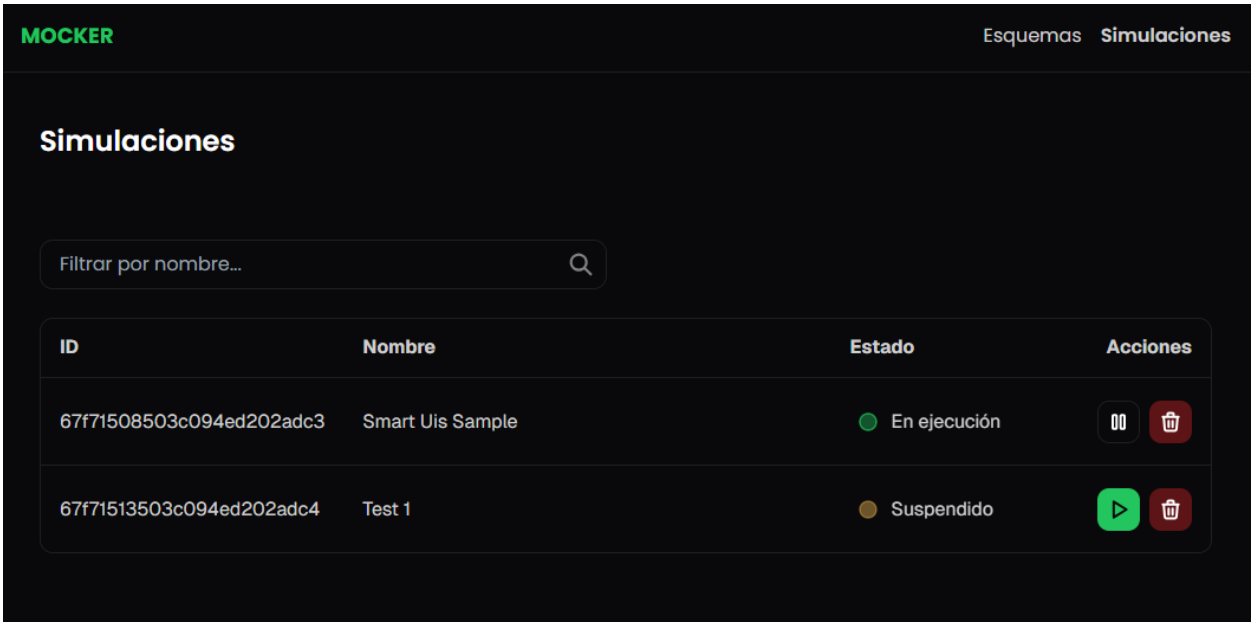
Estos comandos permiten una interacción clara e intuitiva con el sistema de simulación desde la línea de comandos, facilitando la gestión de esquemas, simulaciones y su ejecución en distintos entornos (véase Apéndice F para entender la estructura y componentes internos de la CLI).

6.3 Interfaz Web

La interfaz web fue desarrollada utilizando el *framework* Angular, basado en TypeScript, lo cual permitió una estructura modular, escalable y mantenible para la gestión de las simulaciones (véase Figura 19).

Figura 19

Vista de la interfaz web para la gestión de simulaciones



Nota: Captura de pantalla de la interfaz web, mostrando la vista “Simulaciones”, donde se muestra el listado de simulaciones, su estado actual y los botones de control para su gestión.

6.3.1 Arquitectura

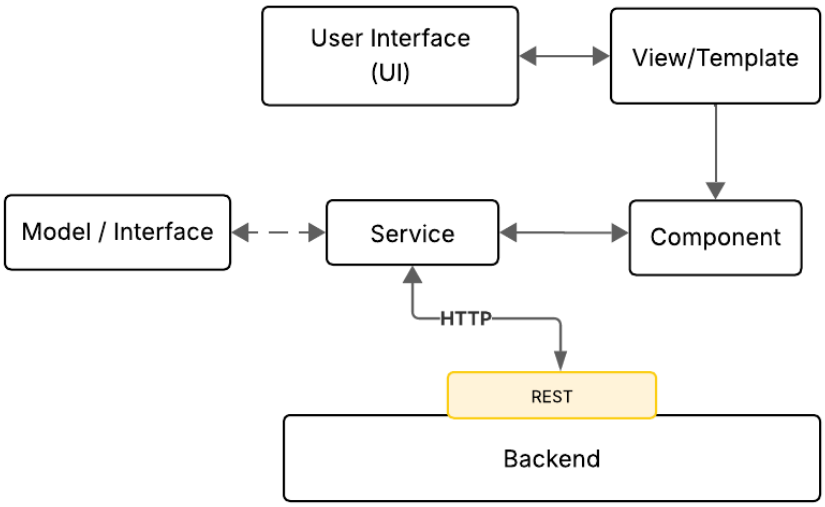
La aplicación Angular sigue una arquitectura basada en **componentes y servicios**, que permite una clara separación de responsabilidades:

- **Componentes:** Encargados de la representación visual y del manejo de eventos del usuario en cada una de las vistas (Esquemas, Simulaciones y Logs).
- **Servicios:** Responsables de gestionar la lógica que permite la interacción entre la interfaz web con el *backend* a través de solicitudes HTTP hacia la API REST.

Esta organización facilita el mantenimiento, la reutilización de código y la extensión del sistema (véase Figura 20).

Figura 20

Flujo de arquitectura de aplicación Angular



Esta organización facilita el mantenimiento, la reutilización de código y la extensión del sistema (véase Apéndice G para más detalles sobre la organización modular y la arquitectura de la interfaz).

6.3.2 Vistas Principales

La interfaz se compone de tres vistas clave:

- **Esquemas:** Permite al usuario cargar archivos de configuración que definen los parámetros de simulación. El sistema realiza validaciones iniciales y persiste el esquema en la base de datos a través del *backend*.
- **Simulaciones:** Muestra una tabla con las simulaciones existentes, permitiendo iniciar, pausar o eliminar una simulación. Se visualiza su estado actual (en ejecución o suspendida) y se pueden ejecutar acciones en tiempo real.
- **Logs:** Muestra en tiempo real los mensajes generados por las simulaciones activas, lo cual permite monitorear y analizar el comportamiento del sistema.

Estas vistas permiten una interacción clara e intuitiva con el sistema de simulación, facilitando la gestión de esquemas, simulaciones y su monitoreo (véase Apéndice H para visualizar los mockups de la interfaz).

6.3.3 Comunicación con el Backend

La interfaz web se comunica con el *backend* mediante una API REST utilizando solicitudes HTTP. Esta interacción es gestionada por los servicios de Angular, que envían datos al servidor y procesan las respuestas. Así, se permite la persistencia de esquemas, el control de simulaciones y la visualización en tiempo real de los logs, manteniendo una separación clara entre la lógica de presentación y la lógica del sistema.

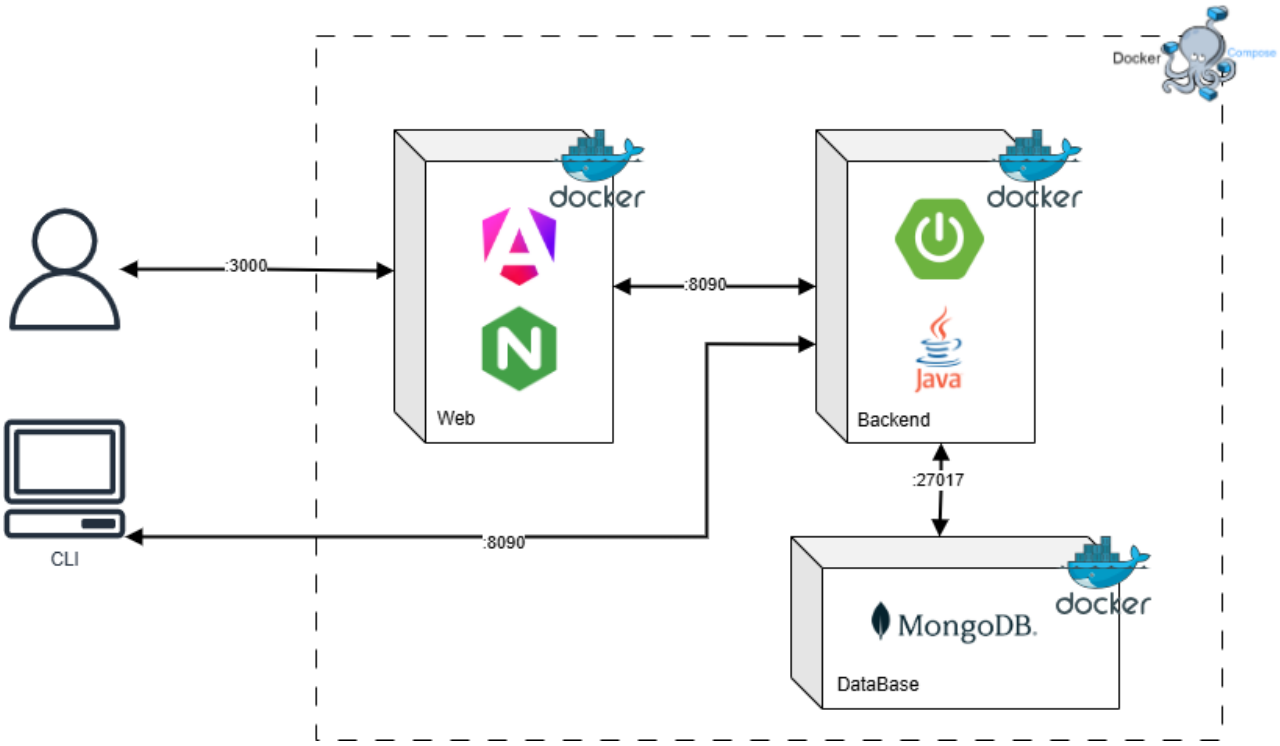
6.4 Despliegue

La arquitectura de despliegue del sistema se basa en contenedores de Docker organizados en un único host. El sistema está compuesto por tres servicios principales: web, *backend* y la base de datos. Cada uno de estos servicios se ejecuta en su propio contenedor, facilitando la independencia entre componentes y simplificando el mantenimiento (véase Figura 21).

El servidor expone una API REST en el puerto 8090, la cual es consumida tanto por la interfaz web como por un cliente de consola (CLI) utilizado para pruebas locales y automatización. El servidor también se conecta a la base de datos MongoDB a través del puerto 27017, almacenando allí los escenarios de simulación, configuraciones.

Figura 21

Diagrama de Despliegue.



Una vez completado el proceso de implementación, es fundamental verificar el correcto funcionamiento de la solución propuesta y su desempeño bajo diferentes condiciones de uso. En este sentido, el siguiente capítulo se centra en la validación de la solución, mediante la ejecución de pruebas funcionales y de rendimiento.

El proyecto completo, incluyendo los archivos de configuración, scripts de despliegue y código fuente de los servicios, se encuentra disponible en el repositorio oficial en GitHub (véase Apéndice I para acceder al repositorio y su estructura).

7 Validación de la Solución

La validación de la solución se llevó a cabo con el objetivo de garantizar que los componentes desarrollados cumplieran con los requisitos funcionales y no funcionales establecidos en las fases de análisis y diseño. Se realizaron pruebas orientadas a verificar el correcto funcionamiento del sistema de simulación de sensores en condiciones controladas, evaluando tanto su comportamiento lógico como su capacidad de respuesta frente a distintas entradas del usuario. Una descripción detallada de la ejecución de pruebas puede consultarse en el Apéndice J.

7.1 Plan de Pruebas

El plan de pruebas definido para esta solución contempla la verificación de funcionalidades clave, asegurando la coherencia y robustez del sistema en todos sus componentes: web, *backend*, CLI y base de datos. Las pruebas se diseñaron para validar los escenarios más representativos de uso por parte de los usuarios finales y desarrolladores. La Tabla 10 resume los casos de prueba definidos.

Tabla 10

Plan de pruebas.

ID	Objetivo	Resultado esperado
P-01	Crear y visualizar un esquema desde la interfaz Web.	El esquema se almacena en MongoDB y aparece en la lista de esquemas.
P-02	Crear y visualizar un esquema desde la interfaz de línea de comandos	El esquema se almacena en MongoDB y aparece en la lista de esquemas.

P-03	Crear, iniciar y visualizar una simulación.	Simulación visible en la lista y controlable desde la interfaz web y CLI.
P-04	Visualizar logs en tiempo real de una simulación.	Los logs se actualizan en la vista sin necesidad de recargar.
P-05	Validar entradas incorrectas o incompletas en la creación de esquemas.	El sistema detecta errores como: nombres vacíos, configuraciones inválidas, duplicados o ausencia de elementos esenciales (como generadores), mostrando mensajes de error y bloqueando la acción.
P-06	Validar el rendimiento del sistema frente a escenarios de simulación.	El sistema mantiene un uso de CPU y memoria dentro de márgenes aceptables, incluso durante la ejecución de simulaciones simultáneas, sin afectar la estabilidad.

7.2 Escenarios

Con el objetivo de validar el comportamiento del simulador y demostrar la capacidad de configuración del lenguaje de dominio específico, se definieron tres escenarios de simulación con niveles progresivos de complejidad: básico, intermedio y avanzado. Cada uno de estos escenarios busca reflejar diferentes condiciones de uso, desde una validación básica hasta una simulación más cercana a un entorno de producción real. La Tabla 11 describe cada uno de estos escenarios.

Tabla 11

Escenarios definidos.

Escenario	Descripción
Básico	Uso de un protocolo con un único generador y muestreo básico (Figura 56 del Apéndice J).
Intermedio	Uso simultáneo de MQTT y AMQP, Incluye múltiples generadores y estrategias de muestreo mixtas (Figura 57 del Apéndice J).
Avanzado	Configuración completa con generadores basados en distribuciones estadísticas y muestreadores con múltiples pasos (Figura 58 del Apéndice J).

7.3 Ambiente de Pruebas

Se implementó un ambiente de pruebas basado en infraestructura en la nube, diseñado para evaluar el comportamiento del sistema en condiciones de recursos limitados, lo cual simula un escenario de implementación inicial realista. Dicho ambiente se configuró utilizando una instancia EC2 de Amazon Web Services con las siguientes especificaciones:

Tabla 12

Especificaciones de la Máquina Virtual

Recurso	Valor
Tipo de instancia	t2.micro
CPU	1 vCPU
Memoria RAM	1 GB
Sistema operativo	Ubuntu Server 24.04 LTS

7.4 Ejecución de las Pruebas

La ejecución de las pruebas se llevó a cabo en un entorno controlado y reproducible, con todos los componentes del sistema desplegados mediante contenedores Docker. Esta configuración permitió mantener condiciones constantes durante todas las evaluaciones, garantizando así la validez de los resultados y facilitando la identificación de comportamientos específicos del sistema bajo distintos niveles de carga.

Cada componente de la solución fue evaluado utilizando herramientas apropiadas para su interfaz. El *backend* fue probado mediante la herramienta curl, enviando solicitudes HTTP con distintas configuraciones de entrada y analizando las respuestas obtenidas tanto en su contenido como en los códigos de estado HTTP, incluyendo casos exitosos y errores controlados. La interfaz web fue sometida a pruebas manuales que simularon flujos típicos de usuario, como la creación de esquemas, el inicio de simulaciones y la visualización de resultados. Por su parte, la CLI fue validada a través de la ejecución directa de comandos desde la terminal, verificando su integración con el *backend* y su cumplimiento con los criterios funcionales definidos.

Para las pruebas de rendimiento (P-06), se diseñaron escenarios específicos que incluyeron la ejecución simultánea de múltiples simulaciones. Por cada uno de los tres escenarios de simulación definidos (básico, intermedio y avanzado), se realizaron ejecuciones con 5, 10, 15 y 20 simulaciones concurrentes. Estas pruebas tuvieron como objetivo observar:

- Uso promedio y pico de CPU y memoria RAM
- El comportamiento general del sistema bajo diferentes niveles de carga.
- La ocurrencia de fallos, bloqueos o condiciones de inestabilidad.

Este enfoque permitió evaluar la capacidad del sistema para escalar ante cargas progresivas, identificar posibles cuellos de botella en su arquitectura, y validar su desempeño en condiciones que simulan un entorno real de producción.

Durante todo el proceso de validación, se recopilaron registros del sistema, trazas de ejecución, salidas por consola y capturas de pantalla, los cuales respaldan los resultados presentados (véase Apéndice J). Además, el entorno de pruebas fue monitoreado de forma continua mediante una instancia EC2 de AWS, lo que permitió observar en tiempo real el consumo de recursos y descartar interferencias externas, asegurando una evaluación objetiva y confiable del desempeño del sistema.

Una vez completadas las pruebas funcionales y de rendimiento bajo los distintos escenarios, se procedió a la recopilación, análisis y consolidación de los resultados obtenidos. En la siguiente sección, se presentan dichos resultados organizados por escenario, junto con el análisis de los comportamientos observados, tanto desde el punto de vista funcional como de rendimiento.

7.5 Resultados de las Pruebas

Con base en la ejecución de las pruebas funcionales y de rendimiento descritas en la sección anterior, se recopilaron y analizaron los datos obtenidos para evaluar la estabilidad, capacidad de respuesta y correcto funcionamiento de la solución desarrollada. Esta sección presenta los resultados de forma estructurada, incorporando métricas de rendimiento, estadísticas de ejecución y observaciones relevantes, organizadas por tipo de escenario.

7.5.1 Resultados de Pruebas Funcionales

Las pruebas funcionales se llevaron a cabo de forma manual mediante la interacción directa con la interfaz web, la línea de comandos (CLI) y el backend por medio de solicitudes HTTP. Cada prueba fue repetida tres veces por escenario (básico, intermedio y avanzado), con el fin de

garantizar consistencia y detectar posibles comportamientos intermitentes. La siguiente tabla resume los resultados obtenidos:

Tabla 13

Resultados de pruebas funcionales.

ID	Resultado	Observación
P-01	Éxito	El esquema se creó exitosamente desde la web, se almacenó en MongoDB y fue visible en la lista.
P-02	Éxito	El esquema se creó exitosamente desde la CLI, se almacenó en MongoDB y fue visible en la lista.
P-03	Éxito	La interfaz web y CLI permitieron crear, iniciar y detener simulaciones.
P-05	Éxito	Los logs se visualizaron en tiempo real desde la interfaz Web.

Observación: No se presentaron fallos funcionales, sin embargo, se identificó una mejora potencial en la usabilidad de la CLI al crear múltiples esquemas.

7.5.2 Resultados de Pruebas de Rendimiento

Con el objetivo de evaluar el rendimiento del sistema propuesto, se diseñaron pruebas específicas orientadas a ejecutar múltiples simulaciones de forma simultánea. Estas pruebas se estructuraron en función de los tres escenarios definidos previamente (básico, intermedio y avanzado), ejecutando lotes compuestos por 5, 10, 15 y 20 simulaciones paralelas. La finalidad de este enfoque fue observar el comportamiento del sistema bajo distintas cargas de trabajo y analizar

métricas clave como el uso de CPU, consumo de memoria RAM, estabilidad en la generación de datos y tiempos de respuesta en entornos concurrentes.

A continuación, se presentan los resultados obtenidos a partir de estas pruebas, organizados por escenario.

7.5.3 Escenario de simulación básico

En el escenario básico, el consumo de recursos se mantuvo bastante estable y bajo incluso a medida que aumentaba la cantidad de simulaciones. El uso de CPU creció de forma muy leve, pasando de 0.06 % con cero simulaciones a solo 0.15 % con 20 simulaciones, como se muestra en la Figura 22. Del mismo modo, la memoria RAM mostró un incremento gradual, de 107.9 MiB a 123.5 MiB, reflejando un crecimiento proporcional a las simulaciones (Figura 23).

Figura 22

Uso de CPU para el Escenario Básico.

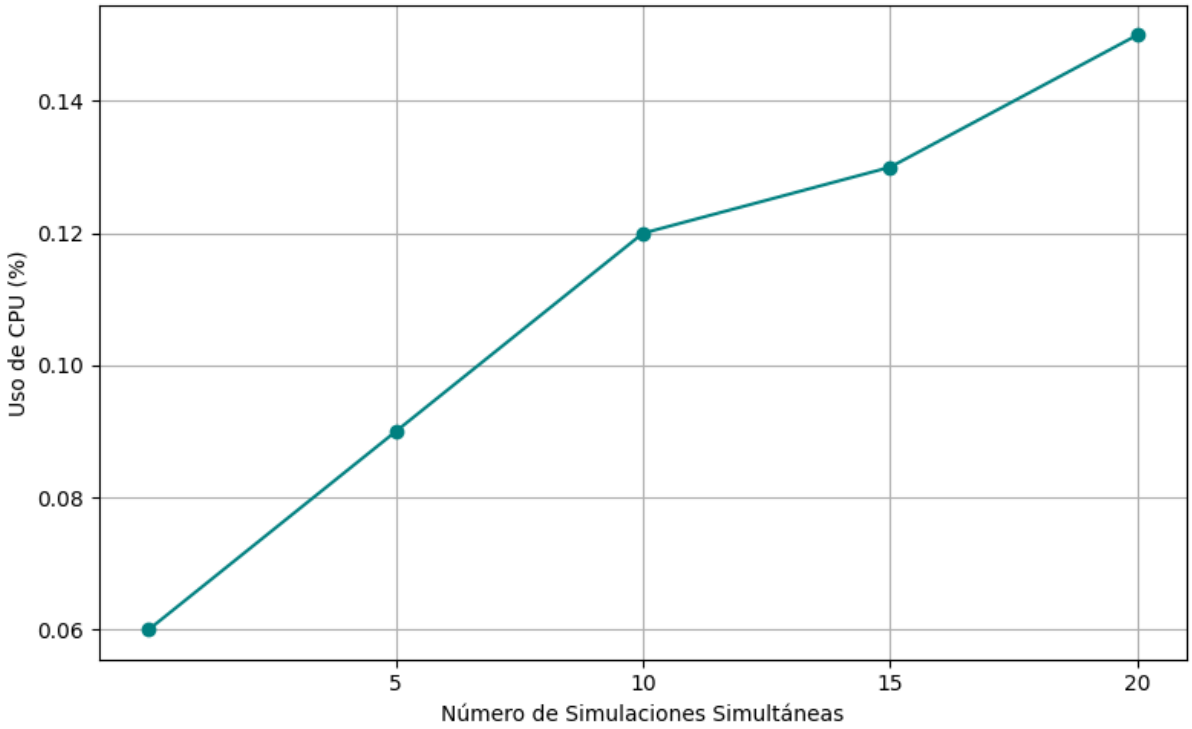
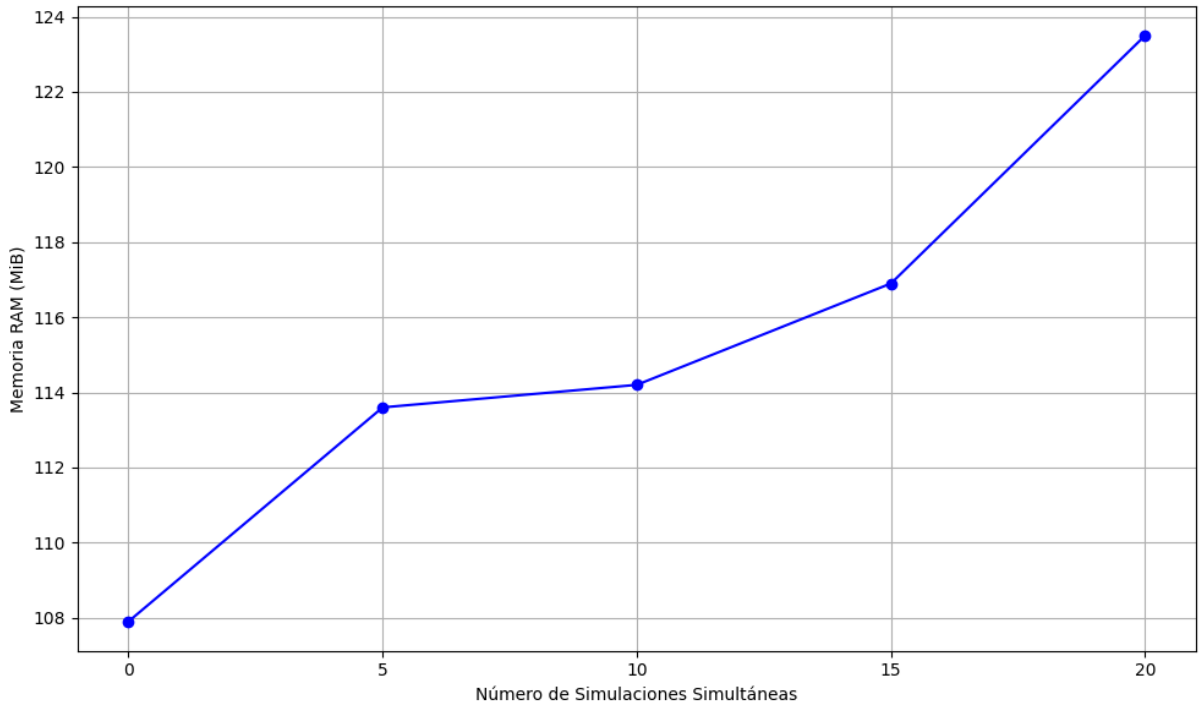


Figura 23

Consumo de Memoria RAM para el Escenario Básico.



Nota. MiB (Mebibyte) equivale a 1,048,576 bytes. Se diferencia de MB (Megabyte), que equivale a 1,000,000 bytes.

7.5.4 Escenario de simulación intermedio

En el escenario intermedio se observó una escalada más pronunciada tanto en el uso de CPU como en la memoria RAM. La CPU pasó de un 0.08 % en estado inactivo a un 0.92 % al ejecutar 20 simulaciones, con aumentos constantes en cada punto intermedio (Figura 24). En cuanto a la memoria, el crecimiento fue más notable, partiendo de 106.4 MiB y alcanzando 132.5 MiB, como se observa en la Figura 25.

Figura 24

Uso de CPU para el Escenario Intermedio.

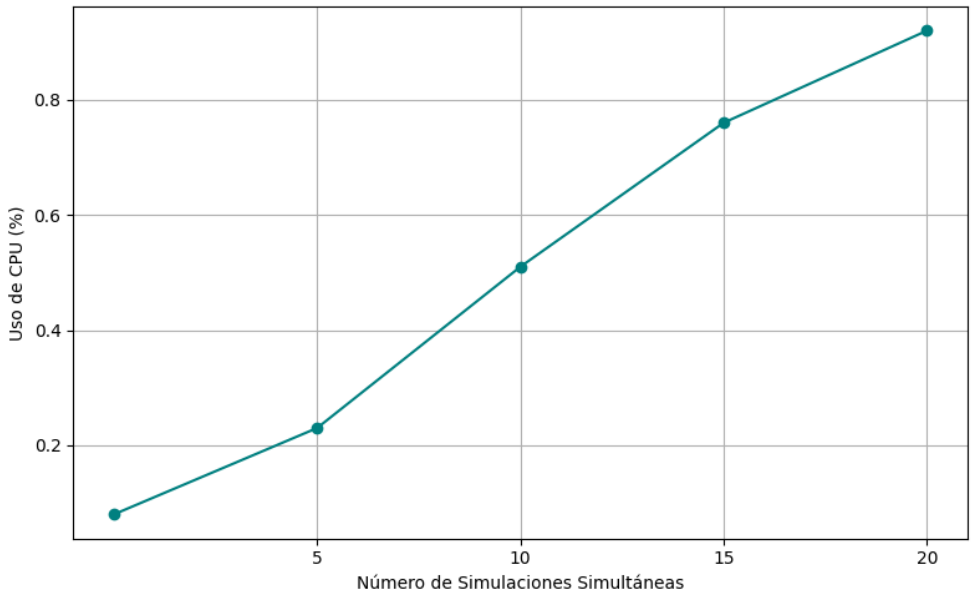
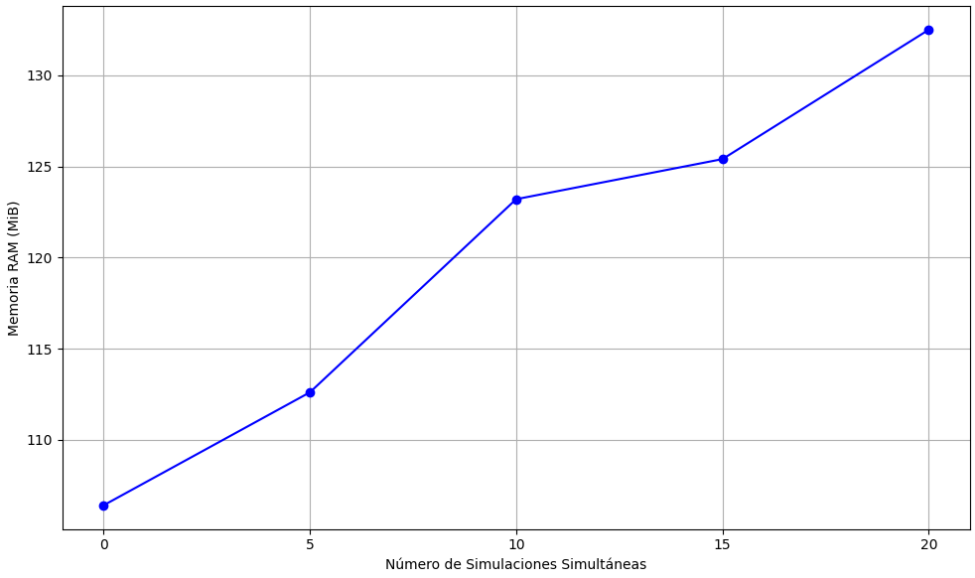


Figura 25

Consumo de Memoria RAM para el Escenario Intermedio.



Nota. MiB (Mebibyte) equivale a 1,048,576 bytes. Se diferencia de MB (Megabyte), que equivale a 1,000,000 bytes.

7.5.5 Escenario de simulación Avanzado

El escenario avanzado presentó un comportamiento más exigente en términos de recursos. El uso de CPU mostró un crecimiento consistente y más agresivo, comenzando en 0.09 % con cero simulaciones y alcanzando 0.75 % con 20 simulaciones (Figura 26). Aunque el consumo de CPU fue inferior al del escenario intermedio a 20 simulaciones, el comportamiento fue más irregular y exigente en los puntos intermedios. La memoria RAM, por su parte, aumentó desde 110.2 MiB hasta 134 MiB, manteniendo una tendencia de crecimiento moderado (Figura 27).

Figura 26

Uso de CPU para el Escenario Avanzado.

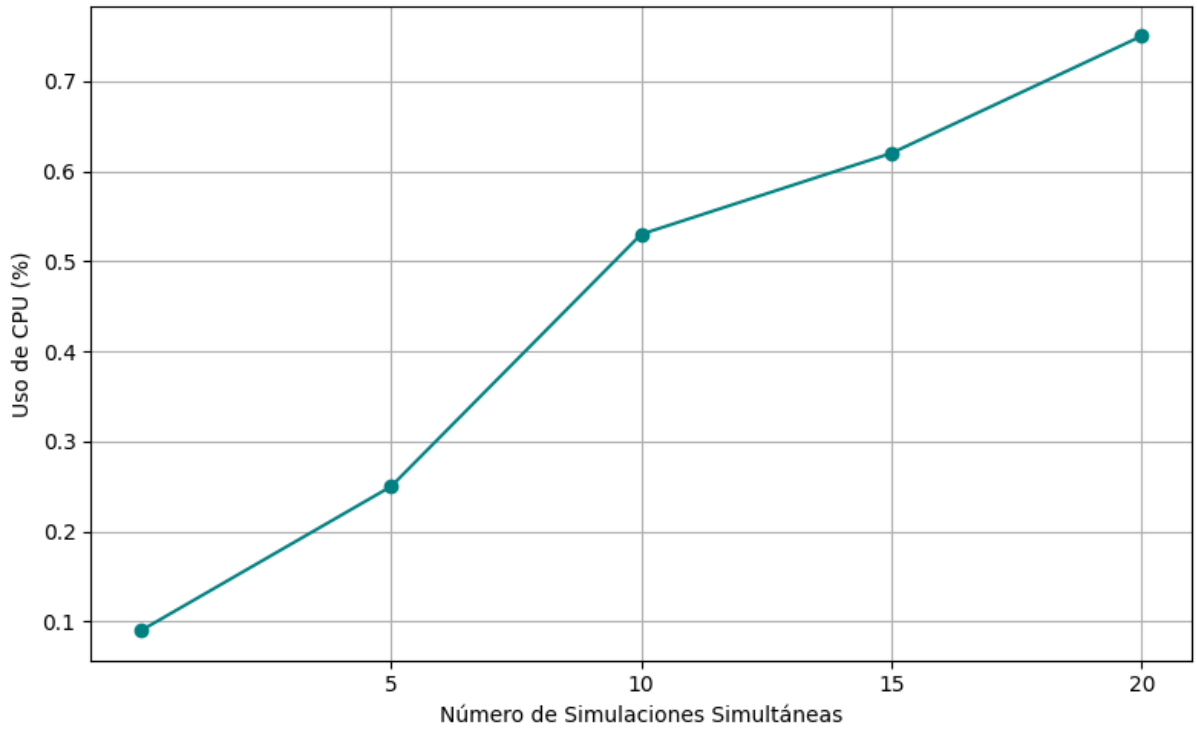
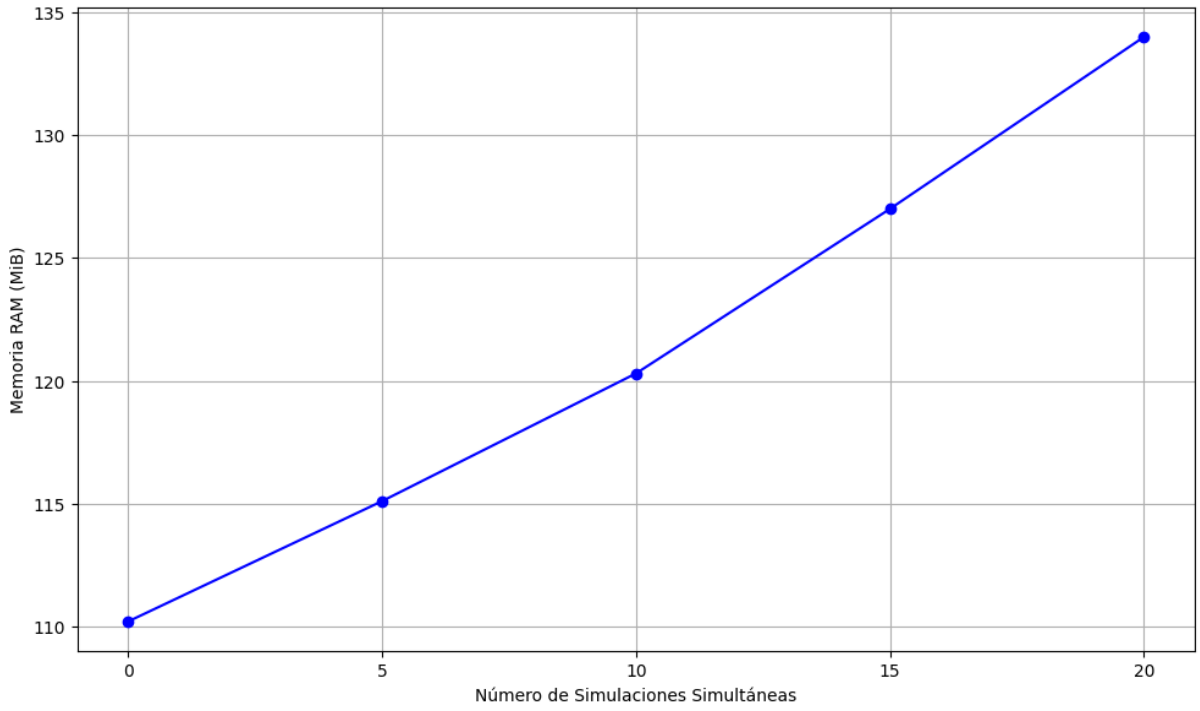


Figura 27

Consumo de Memoria RAM para el Escenario Avanzado.



Nota. MiB (Mebibyte) equivale a 1,048,576 bytes. Se diferencia de MB (Megabyte), que equivale a 1,000,000 bytes.

7.5.6 Análisis de resultados

Los resultados obtenidos a partir de las pruebas funcionales y de rendimiento realizadas al sistema permiten establecer que la solución desarrollada cumple con los objetivos propuestos. Desde el punto de vista funcional, el sistema demostró ser capaz de interpretar correctamente los archivos de configuración en formato YAML, permitiendo la definición estructurada y flexible de escenarios de simulación de sensores. Se verificó la correcta ejecución de las simulaciones, así como la generación y transmisión de datos sintéticos conforme a los parámetros establecidos, validando así la coherencia lógica del sistema y su adherencia al diseño propuesto.

En cuanto a las pruebas de rendimiento, se evaluó el comportamiento del sistema bajo diferentes niveles de carga (básico, intermedio y avanzado), considerando el consumo de recursos de CPU y memoria en cada caso. Los resultados evidencian un aumento progresivo en el uso de recursos conforme al número de simulaciones concurrentes, comportamiento esperado en sistemas de esta naturaleza. No obstante, dicho crecimiento se mantuvo dentro de márgenes aceptables, sin comprometer la estabilidad ni la disponibilidad del sistema, lo cual da cuenta de una arquitectura eficiente y escalable.

En términos generales, la solución propuesta no solo es funcional y operativamente robusta, sino también eficiente en términos de consumo de recursos. Esto la convierte en una herramienta adecuada para entornos de prueba, investigación y desarrollo de sistemas inteligentes. Además, se establece una base sólida para futuras mejoras y extensiones orientadas a escenarios más complejos o de mayor escala.

8 Trabajo Futuro

El proyecto está diseñado con un enfoque extensible, por eso existen diversas líneas de trabajo futuro que puedan fortalecer y ampliar su funcionalidad. Una de las mejoras potenciales es la definición de los esquemas de manera gráfica, lo que permitiría a los usuarios configurar simulaciones de manera visual e intuitiva, sin necesidad de modificar archivos YAML manualmente. Para ello, se podría desarrollar una interfaz que genere automáticamente los archivos de descripción de la simulación a partir de formularios dinámicos, facilitando la personalización y reduciendo la posibilidad de errores.

Adicionalmente, la incorporación de un sistema de visualización de datos lo cual podría mejorar la interpretación y el análisis de resultados obtenidos en las simulaciones. Se podrían incluir *dashboards* interactivos con gráficos en tiempo real, herramientas para filtrar y comparar datos, entre otros. Estas nuevas funcionalidades facilitarían la identificación de patrones y tendencias en la información generada.

Para mejorar la interoperabilidad del sistema, podría explorarse la definición e implementación de nuevos protocolos de comunicación, permitiendo integrar el sistema con una mayor variedad de dispositivos IoT y plataformas externas, asegurando una conexión fluida entre tecnologías que utilizan distintos estándares de comunicación. Como resultado el proyecto se adaptaría con mayor facilidad a entornos y expandir su alcance en aplicaciones del mundo real.

Otra línea de desarrollo es la creación de nuevos muestreadores y generadores de datos, capaces de simular escenarios más complejos y cercanos a la realidad. Esto incluiría la generación de datos con patrones más realistas basados en modelos predictivos y técnicas de aprendizaje

automático, permitiendo evaluar el comportamiento del sistema en condiciones más representativas del mundo real.

9 Conclusiones

El proyecto “Diseño de una solución para la definición de escenarios de simulación de sensores en la plataforma Smart Campus UIS” ha demostrado viabilidad mediante la implementación de un sistema para la generación de datos en tiempo real. Las pruebas funcionales y de rendimiento confirmaron que el sistema puede gestionar múltiples simulaciones de forma simultánea, lo que garantiza estabilidad y confiabilidad en la información procesada.

Entre los principales logros del proyecto está la integración y validación de sus tres componentes fundamentales: el *backend*, la interfaz web y la CLI. La interfaz web permite la configuración e inicio de simulaciones de manera intuitiva, la CLI facilita su gestión por medio de comandos y el *backend* asegura la correcta interpretación de archivos YAML, la gestión de estados de simulación y la persistencia de datos en la base de datos sin errores.

Un aspecto importante del sistema desarrollado es su enfoque intuitivo, ya que permite la definición de simulaciones mediante un Lenguaje de Dominio Específico (DSL) declarativo basado en YAML. Este lenguaje permite a los usuarios definir de forma estructurada elementos como protocolos de comunicación, el tipo de muestreo y generadores de datos, simplificando significativamente la experiencia del usuario al momento crear y personalizar escenarios de simulación sin requerir conocimientos avanzados de programación.

El sistema se caracteriza por su flexibilidad, ya que soporta una amplia variedad de configuraciones. Permite el uso de distintos protocolos de comunicación como MQTT y AMQP, diversos tipos de muestreo y generadores de datos. Esta capacidad de adaptación permite modelar diferentes escenarios de simulación, con un alto grado de personalización.

Las pruebas de rendimiento realizadas al sistema confirmaron su estabilidad y escalabilidad. Se evaluó el consumo de memoria y CPU bajo diferentes niveles de carga, demostrando que la arquitectura manejar múltiples simulaciones concurrentes sin afectar de manera significativa el rendimiento, incluso con 20 simulaciones simultaneas, el sistema mantuvo tiempos de respuesta adecuados, sin indicios de bloqueos o fugas de memoria, validando la eficiencia de recursos utilizados.

En cuanto a los objetivos planteados, se logró cumplir con los requerimientos establecidos, garantizando que los usuarios pudieran configurar, ejecutar y monitorear simulaciones de manera sencilla y confiable. Además, se eliminó la necesidad de invertir en prototipos físicos o desarrollos personalizados para cada escenario, lo que conlleva a una reducción significativa de costos y tiempos durante las etapas de diseño, validación y pruebas.

El proyecto también evidencio la importancia de implementar una arquitectura modular y extensible, ya que facilita la incorporación de nuevos protocolos, generadores y muestreadores, sin afectar el funcionamiento del sistema actual. De igual forma se identificaron oportunidades de mejora, la incorporación de técnicas avanzadas de procesamiento de datos para la detección de anomalías y el desarrollo de una interfaz más intuitiva para la construcción de esquemas mediante formularios dinámicos, reduciendo la necesidad de editar archivos YAML manualmente.

Finalmente, el proyecto tiene gran potencial de crecimiento y evolución. La incorporación de las mejoras mencionadas en el apartado de trabajo futuro no solo agregaría mayor valor a la plataforma, sino que también la consolidaría como una herramienta clave para la investigación, prueba y validación de sistemas IoT, tanto en el entorno del Smart Campus UIS como en diversos campos y aplicaciones relacionadas.

Referencias

Alharbi, E. (2021). People-centric smart campus. *2021 International Symposium on Computer Science and Intelligent Controls (ISCSIC)*, 264-267. https://doi.org/10.1109/ISCSIC54682.2021.00055

Amazon Web Services. (n.d.). ¿Qué es una CLI? *AWS*. Recuperado en 2025 de https://aws.amazon.com/es/what-is/cli/

Arista Networks. (nd.). *EOS 4.33.2F - Command-Line Interface (CLI)*. Recuperado de https://www.arista.com/en/um-eos/eos-command-line-interface-cli

Ashton, K. (2009). That 'Internet of Things' thing. *RFID Journal*. Recuperado de https://www.rfidjournal.com/expert-views/that-internet-of-things-thing/73881/

Bhimani, P., & Panchal, G. (2018). Message delivery guarantee and status update of clients based on IoT-AMQP. En Y. C. Hu, S. Tiwari, K. Mishra, & M. Trivedi (Eds.), *Intelligent communication and computational technologies* (pp. 19-29). Springer. https://doi.org/10.1007/978-981-10-5523-2_2

Coccoli, M., Guercio, A., Maresca, P., & Stanganelli, L. (2014). Smarter universities: A vision for the fast changing digital era. *Journal of Visual Languages & Computing, 25*(6), 1003–1011. https://doi.org/10.1016/j.jvlc.2014.09.007

Hashimyar, M. E., Aiash, M., Khoshkholghi, A., & Nalli, G. (2025). Signature-Based Security Analysis and Detection of IoT Threats in Advanced Message Queuing Protocol. *Network*, *5*(1), 5. https://doi.org/10.3390/network5010005

IBM. (n.d.). Message brokers. *IBM Think*. Recuperado en 2025 de https://www.ibm.com/think/topics/message-brokers

Imbar, R. V., Supangkat, S. H., & Langi, A. Z. R. (2021). Development of smart campus model. *2021 International Conference on ICT for Smart Society (ICISS)*, 1-5. https://doi.org/10.1109/ICISS53185.2021.9533223

Imbar, R. V., Supangkat, S. H., Langi, A. Z. R., & Arman, A. A. (2023). Measurement of campus smartness: The development of smart campus model. *2023 10th International Conference on ICT for Smart Society (ICISS)*, 1-6. https://doi.org/10.1109/ICISS59129.2023.10291750

InnovacionTech. (n.d.). Sensores en el IoT: Papel e importancia en el mundo conectado. Recuperado de https://innovaciontech.com/sensores-en-el-iot-papel-e-importancia-en-el-mundo-conectado/

Internet Society. (2015). La Internet de las cosas—una breve reseña. *Internet Society*. Recuperado de https://www.internetsociety.org/report-InternetOfThings-20160817-es-1

JetBrains. (n.d.). Domain-specific languages: Concepts. *JetBrains*. Recuperado en 2025 de https://www.jetbrains.com/es-es/mps/concepts/domain-specific-languages/

Kenny, T. (2005). Sensors fundamentals. En J. Wilson (Ed.), *Sensor technology handbook* (pp. 1-15). Newnes.

MacCaw, A. (2011). *JavaScript web applications* (1.ª ed.). O'Reilly Media. ISBN 978-1-449-30351-8

Milligan, I., & Baker, J. (2017). Introducción a la línea de comandos en Bash. *The programming historian en español*, *1*(1). https://doi.org/10.46430/phes0013

Mustache. (n.d.). Mustache(1) logic-less templates. Recuperado en 2025 de https://mustache.github.io/mustache.1.html

Polin, K., Yigitcanlar, T., Limb, M., & Washington, T. (2023). The making of smart campus: A review and conceptual framework. *Buildings, 13*(4), 891. https://doi.org/10.3390/buildings13040891

Spohn, M. (2022). *On MQTT scalability in the Internet of Things: Issues, solutions, and future directions*. *Journal of Electronics and Electrical Engineering, 4*, Article 1687. https://doi.org/10.37256/jeee.1120221687

Subbarao, V., Srinivas, K., & Pavithr, R. S. (2019). A survey on Internet of Things-based smart, digital green and intelligent campus. *2019 4th International Conference on Internet of Things: Smart Innovation and Usages (IoT-SIU)*, 1-6. https://doi.org/10.1109/IoT-SIU.2019.8777476

Thirupathi, V., & Sagar, K. (2022). A survey on MQTT bridges, challenges, and its solutions. *International Conference on Automation, Computing and Renewable Systems (ICACRS 2022) - Proceedings*, 58–62. https://doi.org/10.1109/ICACRS55517.2022.10029241

Rao, Rakshith & Swamy, S R. (2020). *Review on Spring Boot and Spring Webflux for Reactive Web Development.* 7. https://www.researchgate.net/publication/341151097_Review_on_Spring_Boot_and_Spring_Webflux_for_Reactive_Web_Development

Apéndices

Apéndice A. Requerimientos Funcionales

Tabla 14

Requerimiento funcional 1.

Número	RF 01
Nombre	Subir archivo de definición de esquema en formato YAML. (WEB)
Descripción	El sistema debe permitir al usuario cargar archivos de configuración de simulaciones en formato YAML.
Prioridad	Alta

Tabla 15

Requerimiento funcional 2.

Número	RF 02
Nombre	Visualizar lista de esquemas (WEB)
Descripción	El sistema debe mostrar una tabla en tiempo real que liste todos los esquemas cargados, incluyendo información como ID, nombre y estado.
Prioridad	Alta

Tabla 16

Requerimiento funcional 3.

Número	RF 03
Nombre	Acciones sobre esquemas (WEB)
Descripción	El sistema debe permitir al usuario realizar las siguientes acciones sobre cada esquema listado: • Iniciar simulación. • Cargar archivo de formato de salida. • Eliminar esquema.
Prioridad	Alta

Tabla 17

Requerimiento funcional 4.

Número	RF 04
Nombre	Filtrar esquemas por nombre (WEB)

Descripción	El sistema debe proporcionar una funcionalidad de búsqueda que permita al usuario filtrar los esquemas por nombre.
Prioridad	Media

Tabla 18

Requerimiento funcional 5.

Número	RF 05
Nombre	Visualizar lista de simulaciones (WEB)
Descripción	El sistema debe mostrar una tabla en tiempo real que liste todas las simulaciones en curso o finalizadas, incluyendo información como ID, nombre y estado.
Prioridad	Alta

Tabla 19

Requerimiento funcional 6.

Número	RF 06
Nombre	Acciones sobre simulaciones (WEB)
Descripción	El sistema debe permitir al usuario realizar las siguientes acciones sobre cada simulación listada: • Iniciar/Pausar simulación. • Eliminar simulación.
Prioridad	Alta

Tabla 20

Requerimiento funcional 7.

Número	RF 07
Nombre	Filtrar simulaciones por nombre (WEB)
Descripción	El sistema debe proporcionar una funcionalidad de búsqueda que permita al usuario filtrar las simulaciones por nombre.
Prioridad	Media

Tabla 21

Requerimiento funcional 8.

Número	RF 08
Nombre	Acceso a logs de simulación (WEB)
Descripción	El sistema debe permitir al usuario acceder a los logs detallados de una simulación específica al seleccionar dicha simulación de la lista.

Prioridad	Alta

Tabla 22

Requerimiento funcional 9.

Número	RF 09
Nombre	Visualizar logs en tiempo real (WEB)
Descripción	El sistema debe permitir al usuario visualizar en tiempo real los logs generados por una simulación seleccionada.
Prioridad	Alta

Tabla 23

Requerimiento funcional 10.

Número	RF 10
Nombre	Controles desde la vista de logs (WEB)
Descripción	Desde la interfaz de visualización de logs, el sistema debe permitir al usuario: • Pausar la simulación. • Reanudar la simulación. • Eliminar la simulación.
Prioridad	Media

Tabla 24

Requerimiento funcional 11.

Número	RF 11
Nombre	Configuración de Host y Puerto (CLI)
Descripción	El sistema debe permitir al usuario configurar el host y el puerto del servidor mediante variables de entorno.
Prioridad	Alta

Tabla 25

Requerimiento funcional 12.

Número	RF 12
Nombre	Mostrar Versión (CLI)
Descripción	Permitir que los usuarios vean la versión de la CLI con **mocker --version.**
Prioridad	Alta

Tabla 26

Requerimiento funcional 13.

Número	RF 13
Nombre	Mostrar Ayuda (CLI)
Descripción	El sistema debe permitir al usuario visualizar la ayuda de la herramienta mediante el comando **mocker --help.**
Prioridad	Alta

Tabla 27

Requerimiento funcional 14.

Número	RF 14
Nombre	Gestión de Esquemas (CLI)
Descripción	El sistema debe permitir al usuario administrar esquemas de simulación, incluyendo su creación, listado, eliminación y configuración.
Prioridad	Alta

Tabla 28

Requerimiento funcional 15.

Número	RF 15
Nombre	Crear Esquema (CLI)
Descripción	El sistema debe permitir al usuario crear un nuevo esquema a partir de un archivo YAML mediante el comando **mocker schema create --file <path>.**
Prioridad	Alta

Tabla 29

Requerimiento funcional 16.

Número	RF 16
Nombre	Listar Esquemas (CLI)
Descripción	El sistema debe permitir al usuario visualizar todos los esquemas almacenados ejecutando el comando **mocker schema list.**
Prioridad	Media

Tabla 30

Requerimiento funcional 17.

Número	RF 17

Nombre	Eliminar Esquema (CLI)
Descripción	El sistema debe permitir al usuario eliminar un esquema específico proporcionando su identificador mediante el comando **mocker schema delete --id <schemaId>.**
Prioridad	Media

Tabla 31

Requerimiento funcional 18.

Número	RF 18
Nombre	Definir Plantilla de Esquema (CLI)
Descripción	El sistema debe permitir al usuario establecer una plantilla de esquema a partir de un archivo de texto mediante el comando **mocker schema template --file <path> --id <schemaId>.**
Prioridad	Alta

Tabla 32

Requerimiento funcional 19.

Número	RF 19
Nombre	Gestión de Simulaciones (CLI)
Descripción	El sistema debe permitir al usuario administrar simulaciones, incluyendo su creación, visualización, inicio, detención y eliminación.
Prioridad	Alta

Tabla 33

Requerimiento funcional 20.

Número	RF 20
Nombre	Crear Simulación (CLI)
Descripción	El sistema debe permitir al usuario crear una nueva simulación basada en un esquema existente mediante el comando **mocker simulation create --id <schemaId>.**
Prioridad	Alta

Tabla 34

Requerimiento funcional 21.

Número	RF 21
Nombre	Listar Simulaciones (CLI)

Descripción	El sistema debe permitir al usuario visualizar todas las simulaciones disponibles ejecutando el comando **mocker simulation list.**
Prioridad	Media

Tabla 35

Requerimiento funcional 22.

Número	RF 22
Nombre	Iniciar Simulación (CLI)
Descripción	El sistema debe permitir al usuario iniciar una simulación específica proporcionando su identificador mediante el comando **mocker simulation start --id <simulationId>.**
Prioridad	Alta

Tabla 36

Requerimiento funcional 23.

Número	RF 23
Nombre	Detener Simulación (CLI)
Descripción	El sistema debe permitir al usuario detener una simulación en ejecución proporcionando su identificador mediante el comando **mocker simulation stop --id <simulationId>.**
Prioridad	Alta

Tabla 37

Requerimiento funcional 24.

Número	RF 24
Nombre	Eliminar Simulación (CLI)
Descripción	El sistema debe permitir al usuario finalizar inmediatamente una simulación en ejecución mediante el comando **mocker simulation kill --id <simulationId>.**
Prioridad	Alta

Tabla 38

Requerimiento funcional 25.

Número	RF 25
Nombre	Gestión de Esquemas
Descripción	El sistema debe permitir crear, listar, actualizar y eliminar esquemas mediante una API REST.

Prioridad	Alta

Tabla 39

Requerimiento funcional 26.

Número	RF 26
Nombre	Gestión de Simulaciones
Descripción	El sistema debe permitir crear, listar, iniciar, detener y eliminar simulaciones mediante una API REST.
Prioridad	Alta

Tabla 40

Requerimiento funcional 27.

Número	RF 27
Nombre	Logs de Simulación
Descripción	El sistema debe permitir consultar logs detallados de una simulación específica.
Prioridad	Alta

Tabla 41

Requerimiento funcional 28.

Número	RF 28
Nombre	Transmisión en tiempo real
Descripción	El sistema debe permitir la transmisión en tiempo real del estado de una simulación.
Prioridad	Media

Tabla 42

Requerimiento funcional 29.

Número	RF 29
Nombre	Persistencia
Descripción	El sistema debe almacenar esquemas en una base de datos.
Prioridad	Alta

Apéndice B. Requerimientos No Funcionales

Tabla 43

Requerimiento no funcional 1.

Número	RNF 01
Nombre	Diseño Responsivo
Descripción	El sistema debe tener un Diseño Responsivo de forma que la información se vea de forma agradable para diversos tamaños de pantalla.
Prioridad	Alta

Tabla 44

Requerimiento no funcional 2.

Número	RNF 02
Nombre	Usabilidad
Descripción	El sistema debe contar con una interfaz de usuario intuitiva y fácil de navegar, facilitando la gestión de esquemas, simulaciones y logs.
Prioridad	Alta

Tabla 45

Requerimiento no funcional 3.

Número	RNF 03
Nombre	Rendimiento
Descripción	El sistema debe ser rápido y no bloquearse con el flujo normal de navegación del usuario.
Prioridad	Alta

Tabla 46

Requerimiento no funcional 4.

Número	RNF 04
Nombre	Desempeño
Descripción	El sistema debe manejar múltiples simulaciones concurrentes sin afectar el rendimiento.
Prioridad	Alta

Tabla 47

Requerimiento no funcional 5.

Número	RNF 05
Nombre	Persistencia
Descripción	El sistema debe almacenar datos en MongoDB con índices para mejorar el rendimiento.
Prioridad	Alta

Tabla 48

Requerimiento no funcional 6.

Número	RNF 06
Nombre	Portabilidad
Descripción	El sistema debe poder desplegarse en cualquier entorno compatible con Docker.
Prioridad	Alta

Tabla 49

Requerimiento no funcional 7.

Número	RNF 07
Nombre	Mantenibilidad
Descripción	El código debe seguir buenas prácticas de desarrollo y estar documentado.
Prioridad	Alta

Tabla 50

Requerimiento no funcional 8.

Número	RNF 08
Nombre	Interoperabilidad
Descripción	La API debe seguir estándares REST para facilitar la integración con otros sistemas.
Prioridad	Media

Apéndice C. Capacitación Tecnológica

Con el fin de reforzar los conocimientos adquiridos a lo largo de la carrera y fortalecer las habilidades necesarias para el desarrollo del proyecto, se realizaron varios cursos especializados a

través de plataformas en línea como Udemy y YouTube, así como el estudio de documentación oficial y guías sobre estándares de desarrollo. Estas capacitaciones fueron importantes para aprender y manejar tecnologías ampliamente utilizadas en la industria. A continuación, se presentan los cursos más relevantes:

El curso ***Angular: De cero a experto*** fue fundamental para el desarrollo del *frontend*, proporcionando un aprendizaje profundo sobre Angular. A través de este curso, se adquirieron conocimientos clave sobre la creación de componentes, servicios y módulos reutilizables, así como el consumo de APIs REST. Adicionalmente, se exploró el uso de *standalone components* y otras mejoras que optimizan el rendimiento y la estructura del código, permitiendo construir aplicaciones web robustas y escalables.

Figura 28

Angular: De cero a experto – Edición 2025

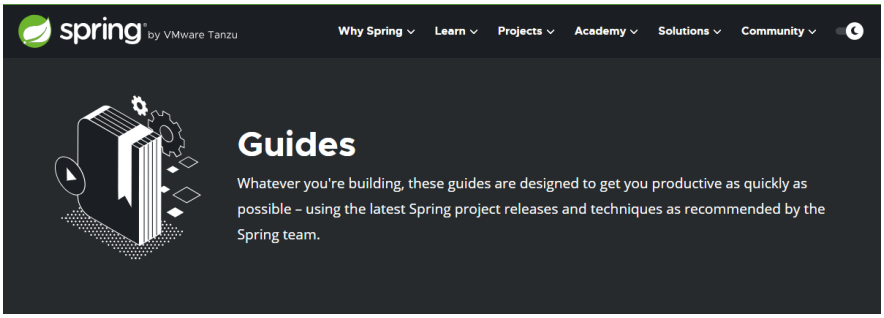
Nota. Tomado de Angular: De cero a experto – Edición 2025. (n.d.). Udemy. https://www.udemy.com/course/angular-fernando-herrera/

Para la implementación del *backend*, los ***Spring Guides*** fueron un recurso fundamental para desarrollar la lógica del servidor con Spring Boot. Estas guías proporcionaron los conocimientos base para la construcción de APIs REST, incluyendo la creación de controladores,

la gestión de dependencias y la configuración de la aplicación, lo que garantiza una estructura modular y escalable.

Figura 29

Spring Guides.

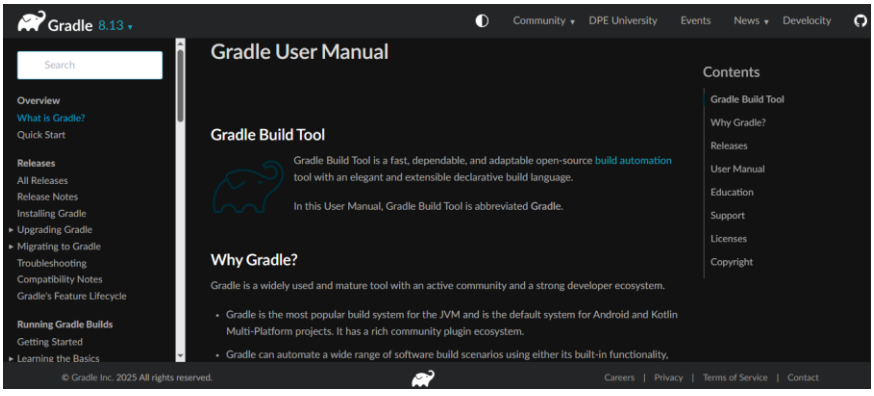


Nota: Tomado de Spring Guides (n.d.). Spring. https://spring.io/guides

En el proceso de desarrollo, la *Gradle User Guide* fue importante para la gestión y automatización del proyecto. Gradle se utilizó como herramienta de construcción para compilar, gestionar dependencias y ejecutar tareas de manera eficiente. Mediante esta guía se adquirieron conocimientos sobre la configuración de *build scripts*, optimización del rendimiento y la adaptación del flujo de trabajo a las necesidades del proyecto.

Figura 30

Gradle user guide.



Nota. Tomado de *Gradle User Guide* (n.d.). Gradle. https://docs.gradle.org/current/userguide/userguide.html

El curso ***Desarrollo Full Stack con Angular y Spring Boot*** en Udemy permitió comprender cómo integrar el *backend* con el *frontend* en una aplicación web. Se abordó la creación de APIs REST con Spring Boot, incluyendo la estructuración del código, el manejo de peticiones HTTP y configuración de seguridad. A demás se profundizó en Angular, enfocándose en la gestión de componentes y servicios para optimizar la comunicación con el *backend*. También se trabajó con herramientas clave como **Node.js, JDK y Git**, fundamentales para un entorno de desarrollo eficiente.

Figura 31

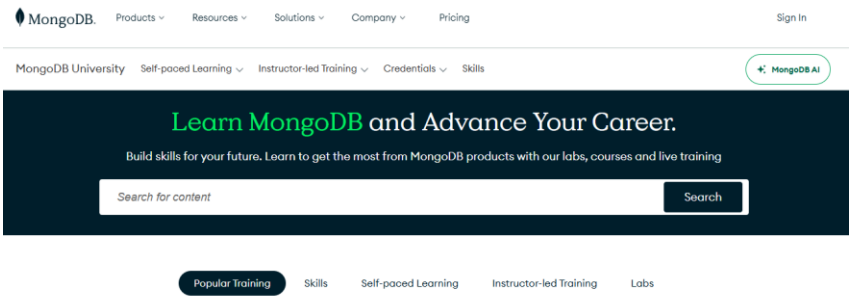
Desarrollo Full Stack integrando Angular, Spring Boot y APIs.

Nota. Tomado de Desarrollo Full Stack integrando Angular, Spring Boot y APIs. (n.d.). Udemy. https://www.udemy.com/course/desarrollo-full-stack-angular-spring-boot-apis/

Para el manejo de datos en el sistema, se utilizó MongoDB, una base de datos NoSQL que permite almacenar información de manera flexible y escalable. A través de la plataforma ***MongoDB Learn,*** se implementaron estrategias para la modelación de datos en formato JSON, la creación de colecciones y ejecución de consultas optimizadas.

Figura 32

Learn MongoDB and Advance Your Career.

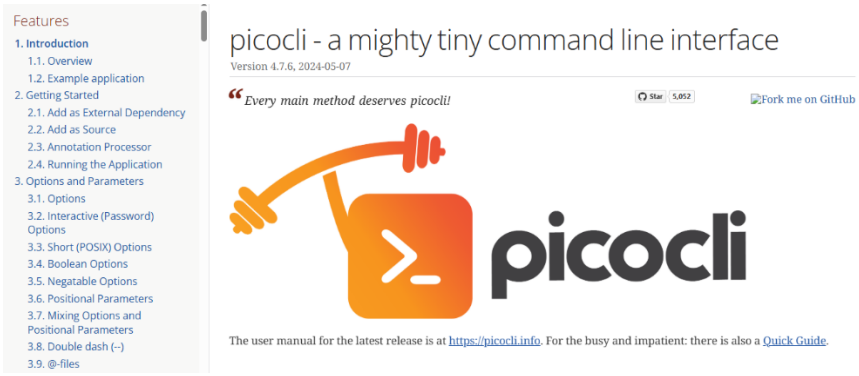


Nota. Tomado de Learn MongoDB and Advance Your Career. (n.d.). MongoDB https://learn.mongodb.com/

La documentación oficial de ***Picocli*** facilitó la implementación de una interfaz de línea de comando (CLI) en el proyecto. A través de estos recursos, se exploraron aspectos como la gestión de argumentos, opciones y subcomandos, lo que permitió estructurar una CLI eficiente y fácil de usar. Además, se trabajó en integración con Java con el fin de mejorar la interacción con el sistema, optimizando la ejecución y configuración de pruebas sin necesidad de interfaces gráficas.

Figura 33

Picocli - a mighty tiny command line interface.



Nota. Tomado de Picocli - a mighty tiny command line interface. (n.d.). Picocli https://picocli.info/

El curso ***Docker Crash Course for Absolute Beginners*** permitió comprender el uso de Docker para el despliegue y gestión de servicios del sistema, con el fin de tener un entorno de ejecución consistente y portable. Se aplicaron conceptos sobre la creación de contenedores, la gestión de imágenes y la orquestación de servicios, lo que facilitó la implementación de la arquitectura del proyecto.

Figura 34

Docker Crash Course for Absolute Beginners.

Nota. Tomado de Docker Tutorial for Beginners (n.d.). TechWorld with Nana. YouTube. https://www.youtube.com/watch?v=pg19Z8LL06w

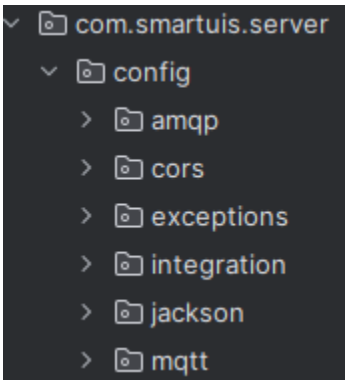
Apéndice D. Desarrollo del backend.

El *backend* del sistema fue desarrollado utilizando Spring Boot, siguiendo su arquitectura tradicional para garantizar modularidad y escalabilidad. Se implementaron servicios REST para la gestión de simulaciones, aprovechando Spring WebFlux para habilitar un procesamiento reactivo y eficiente, permitiendo manejar múltiples solicitudes de manera no bloqueante. Para la simulación, se diseñó una máquina de estados basada en Spring State Machine, lo que permitió

gestionar de manera estructurada las transiciones y estados de cada simulación. Este enfoque facilitó la ejecución controlada de los procesos, asegurando que cada simulación avanzara según las reglas definidas en el DSL, mejorando la trazabilidad y estabilidad del sistema.

Figura 35

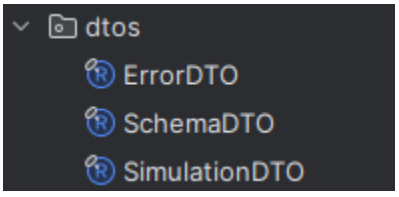
Configuración Global y Protocolos del Sistema



El paquete config contiene todas las configuraciones globales y servicios del sistema, como la configuración de AMQP para mensajería, CORS para control de acceso entre dominios, manejo de excepciones personalizadas, integración con servicios externos, y configuraciones específicas para protocolos como MQTT. Estas configuraciones permiten establecer los parámetros necesarios para que el sistema interactúe de manera eficiente con otros servicios.

Figura 36

Definición de Objetos de Transferencia de Datos

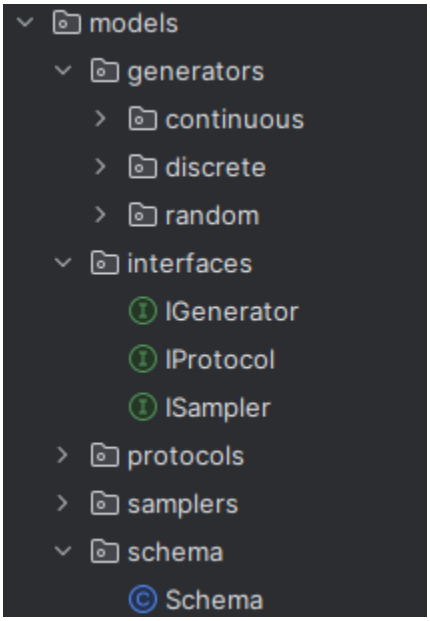


En el paquete dtos se definen los Data Transfer Objects. Los DTOs son clases que se utilizan para la transferencia de datos entre las capas del servidor y los servicios externos. En este

caso, se emplean para definir cómo se estructura la información de las simulaciones y sus componentes.

Figura 37

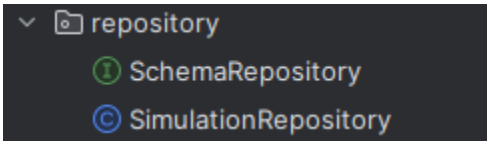
Modelos de Datos y Entidades del Sistema



El paquete models agrupa las clases que representan los objetos de negocio principales de la aplicación, tales como generadores de datos (booleanos, números aleatorios, etc.), protocolos de comunicación, *samplers*, y otros elementos clave del sistema de simulación. Estas clases son la base para manipular y gestionar los datos dentro del servidor.

Figura 38

Gestión de Persistencia y Acceso a Datos

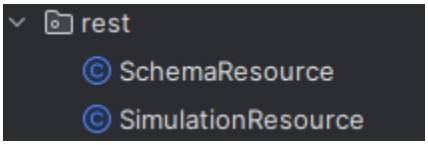


El paquete repository gestiona la persistencia de datos. Contiene las interfaces y clases que interactúan con la base de datos para realizar operaciones de almacenamiento y recuperación de

los datos relacionados con las simulaciones y los esquemas configurados. Asegura que la información se mantenga de manera consistente y accesible.

Figura 39

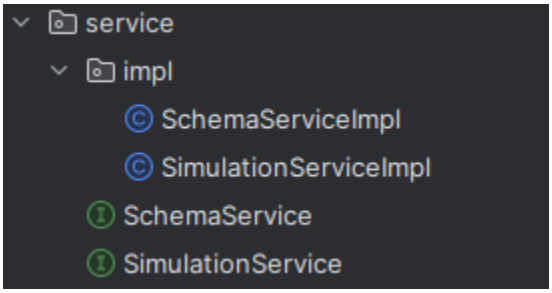
Controladores REST y Gestión de API



En el paquete rest se implementan los controladores REST que exponen las API del sistema. Aquí se gestionan las solicitudes HTTP, permitiendo la comunicación entre el servidor y las aplicaciones cliente (web, CLI). Estos controladores son responsables de manejar las operaciones CRUD, así como de iniciar, detener y obtener información sobre las simulaciones.

Figura 40

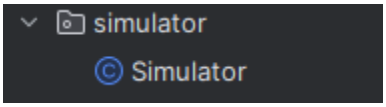
Lógica de Negocio y Gestión de Simulaciones



El paquete service contiene la lógica de negocio. A través de las clases en service/impl, se gestionan las operaciones complejas relacionadas con las simulaciones, como la creación, actualización, y ejecución de estas. Los servicios en este paquete son invocados por los controladores REST y proporcionan los datos y las funcionalidades necesarias.

Figura 41

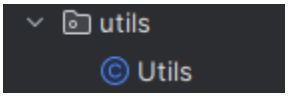
Simulación y Gestión de Estados



En el paquete simulator se encuentran los componentes encargados de ejecutar las simulaciones. Aquí se gestionan los estados de las simulaciones, las máquinas de estado, y las configuraciones dinámicas de cada simulación. Este componente es el núcleo de la funcionalidad de simulación.

Figura 42

Funciones y Herramientas Auxiliares del Proyecto



El paquete utils contiene clases y métodos de utilidad que son utilizados en diversas partes del proyecto. Este paquete incluye funciones generales que ayudan en tareas repetitivas o complejas, como conversión de datos, validaciones, y otros servicios auxiliares.

Apéndice E. Endpoints

Tabla 51

Endpoints para la Gestión de Esquemas y Simulaciones

Método	Endpoint	Descripción
PUT	/api/v1/schema/template/{id}	Actualiza un esquema de simulación utilizando una plantilla predefinida.
GET	/api/v1/schema	Obtiene todos los esquemas de simulación disponibles.

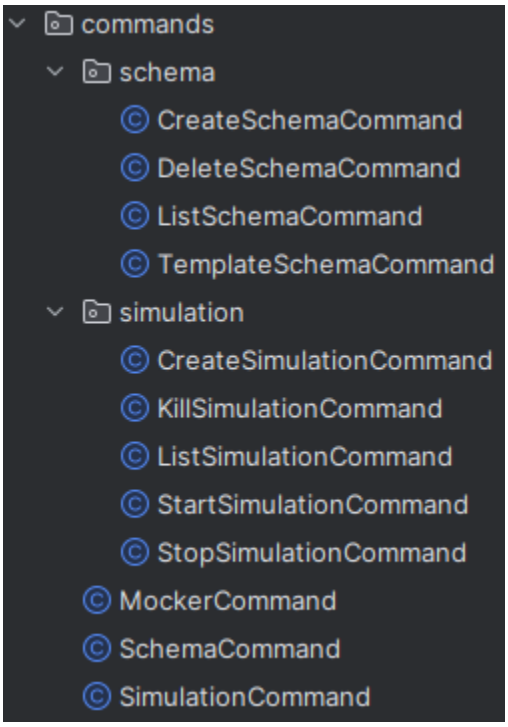
POST	/api/v1/schema	Crea un nuevo esquema de simulación.
GET	/api/v1/schema/{id}	Obtiene un esquema de simulación específico por su id.
DELETE	/api/v1/schema/{id}	Elimina un esquema de simulación específico por su id.
GET	/api/v1/schema/short	Obtiene una lista resumida de los esquemas de simulación.
GET	/api/v1/schema/short/stream	Devuelve los esquemas resumidos de simulación de manera continua (streaming).
GET	/api/v1/simulation	Obtiene todas las simulaciones en curso.
GET	/api/v1/simulation/stream	Devuelve un flujo continuo (streaming) de las simulaciones activas.
GET	/api/v1/simulation/stop/{id}	Detiene una simulación activa identificada por su id.
GET	/api/v1/simulation/start/{id}	Inicia una simulación específica identificada por su id.
GET	/api/v1/simulation/logs/{id}	Recupera los logs de una simulación identificada por su id.
GET	/api/v1/simulation/kill/{id}	Finaliza y elimina una simulación identificada por su id de manera forzada.
GET	/api/v1/simulation/create/{id}	Crea una nueva simulación con el id del schema proporcionado.

Apéndice F. Desarrollo de la CLI

La CLI del sistema fue desarrollada utilizando Picocli junto con Spring Framework, permitiendo una integración fluida con el *backend* y una ejecución eficiente de comandos para la gestión de simulaciones. Gracias a Picocli, se logró una estructura modular y extensible, permitiendo la incorporación de nuevos comandos de manera sencilla. Además, la integración con Spring permitió aprovechar características como la inyección de dependencias y la gestión del ciclo de vida de los procesos, optimizando el rendimiento y la escalabilidad de la aplicación.

Figura 43

Manejo de Comandos

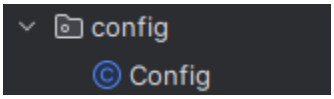


El paquete commands contiene las definiciones de los comandos disponibles en la CLI, que se manejan a través de Picocli. Cada comando es una clase separada que extiende la funcionalidad de Picocli para definir sus parámetros, opciones y cómo se debe ejecutar la acción

relacionada con la simulación. Estos comandos permiten que el usuario controle la simulación mediante la terminal, con opciones detalladas para especificar configuraciones y protocolos.

Figura 44

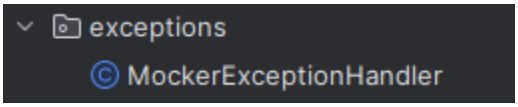
Configuración del Sistema



Este paquete alberga las configuraciones generales necesarias para la integración con el sistema de *backend*. A través de Spring Boot y otras librerías, se gestiona la configuración de los protocolos, la conexión al *backend* y las configuraciones globales para la ejecución de las simulaciones. Asegura que la CLI funcione correctamente al comunicarse con el sistema de simulación.

Figura 45

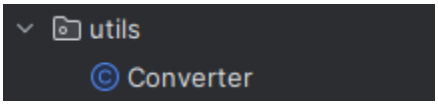
Gestión de Errores



El paquete exceptions maneja los errores y excepciones que pueden surgir durante la ejecución de los comandos en la CLI. Contiene clases personalizadas de excepciones que permiten un control más granular sobre los errores específicos de la aplicación. Además, gestiona los mensajes de error para que el usuario pueda obtener retroalimentación clara sobre cualquier fallo durante la interacción con la simulación. Esto también asegura que las excepciones sean capturadas y procesadas de manera apropiada, manteniendo la estabilidad de la aplicación.

Figura 46

Funciones Auxiliares



El paquete utils contiene diversas utilidades generales y funciones de apoyo que son utilizadas por otras partes de la aplicación. Estas utilidades ayudan a simplificar tareas comunes que se repiten a lo largo del proyecto, como la manipulación de datos, la validación de entradas o la conversión de tipos. Además, puede incluir clases para el manejo de configuraciones o la implementación de lógicas auxiliares que no encajan en la estructura principal del negocio, pero son esenciales para la funcionalidad general de la CLI.

Apéndice G. Desarrollo del Frontend

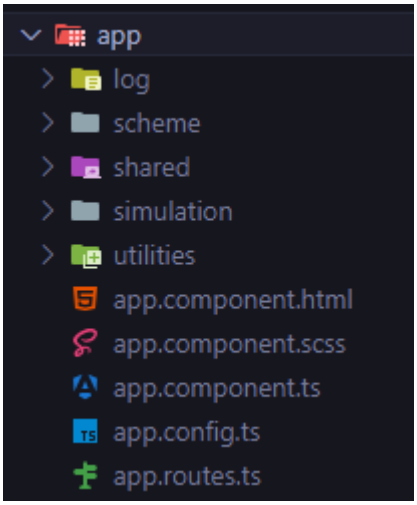
El proyecto está organizado de manera modular, donde la aplicación se divide en distintos módulos para mejorar la escalabilidad y el mantenimiento. La estructura separa responsabilidades, donde la interfaz se desarrolla en componentes, la lógica se gestiona en servicios y las funcionalidades se agrupan en módulos.

AppModule:

Es el punto de entrada de la aplicación, encargándose de su configuración e inicialización. Su diseño permite la integración eficiente de los distintos módulos, asegurando una arquitectura modular y escalable. Además, define las rutas principales y facilita la comunicación entre componentes, proporcionando una base estructurada que optimiza la navegación y el mantenimiento del sistema.

Figura 47

Distribución de archivos AppModule

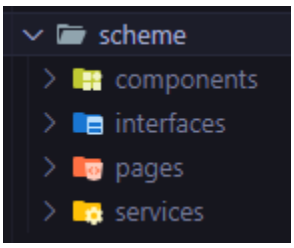


Scheme Module

El módulo de esquemas está diseñado para gestionar y configurar esquemas de simulación dentro del sistema. Mediante una estructura modular que incluye componentes reutilizables, páginas para la administración de esquemas, servicios que manejan la lógica del procesamiento y la comunicación con el *backend*, así como interfaces que garantizan una representación estructurada de los datos.

Figura 48

Distribución de archivos SchemeModule.

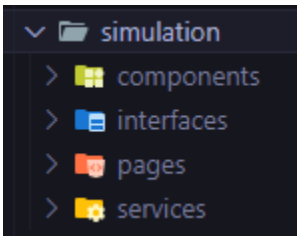


Simulation Module

El módulo de simulación gestiona la ejecución y control de simulaciones dentro del sistema, permitiendo iniciarlas, detenerlas y supervisar su estado. Mediante una estructura modular que incluye componentes reutilizables, páginas para la administración de simulaciones, servicios que manejan la lógica del procesamiento y la comunicación con el *backend* y la comunicación con el *backend*, así como interfaces que garantizan una representación estructurada de los datos.

Figura 49

Distribución de archivos SimulationModule.

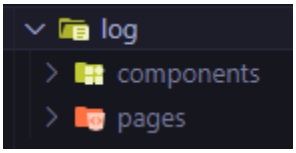


Log Module

El módulo de logs permite la visualización y gestión de los registros generados durante la ejecución de las simulaciones. Mediante una estructura modular que incluye componentes reutilizables para la representación de los logs y páginas para la consulta y análisis, lo que facilita la supervisión del comportamiento de cada simulación en tiempo real.

Figura 50

Distribución de archivos LogModule.

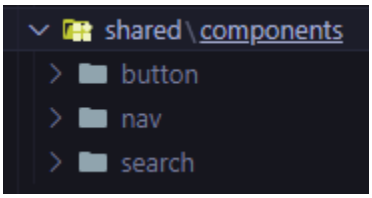


Shared Module

El módulo shared agrupa componentes reutilizables para las vistas de Esquemas y Simulaciones, como la barra de navegación y el filtro de búsqueda. Aunque incluye un botón para la carga de esquemas, este solo se utiliza en la vista de esquemas, quedando disponible para posibles reutilizaciones futuras.

Figura 51

Distribución de archivos SharedModule.



Utilities Module

El módulo de utilidades agrupa los elementos que apoyan la configuración del proyecto, como un archivo de variables SCSS, permitiendo una gestión estructurada y coherente de los estilos en la aplicación.

Enviroments Module

Este módulo gestiona las variables de entorno de la aplicación, donde se definen parámetros como la URL base de la API.

Componentes

Un componente en Angular representa una unidad reutilizable en la interfaz de usuario, compuesta por una plantilla HTML, una hoja de estilos CSS y una clase TypeScript que maneja su lógica y comportamiento. Los componentes en esta aplicación están organizados dentro de cada módulo y se dividen en páginas y componentes reutilizables.

- **Páginas:** Representan vistas completas para funcionalidades específicas, como la gestión de esquemas en SchemeModule y la administración de simulaciones en SimulationModule.
- **Componentes reutilizables:** Son elementos que se emplean en varias vistas, como la barra de navegación y el filtro de búsqueda dentro del SharedModule.
- **Servicios:** Los servicios en Angular encapsulan la lógica de negocio y permiten la comunicación con la API. Su principal función es gestionar la obtención, manipulación y actualización de datos de manera centralizada, evitando la duplicación de código en los componentes.

En esta aplicación, los servicios están organizados según su propósito:

- **Gestión de esquemas (SchemaService):** Se encarga de la carga, creación y eliminación de esquemas. También maneja la conversión de archivos YAML y plantillas, y permite iniciar simulaciones basadas en un esquema.
- **Gestión de simulaciones (SimulationService):** Administra las simulaciones, permitiendo su inicio, detención y eliminación. Además, proporciona acceso a los logs en tiempo real mediante EventSource.

Interfaces

En Angular, las interfaces se utilizan para definir la estructura de los datos que manejan los componentes y servicios, permitiendo tipar objetos y facilitar la comunicación con el *backend*. En esta aplicación, las interfaces se emplean para estandarizar datos como simulaciones y esquemas, asegurando que la información intercambiada entre el *frontend* y el *backend* sigan un formato definido.

Apéndice H. Diseño Mockups

Este apéndice presenta los mockups de las vistas principales del sistema, incluyendo las secciones de Esquemas, Simulaciones y Logs. Estos diseños permiten visualizar la disposición de los elementos en la interfaz y sirven como guía para la implementación del *frontend.*

Figura 47

Vista de Esquemas

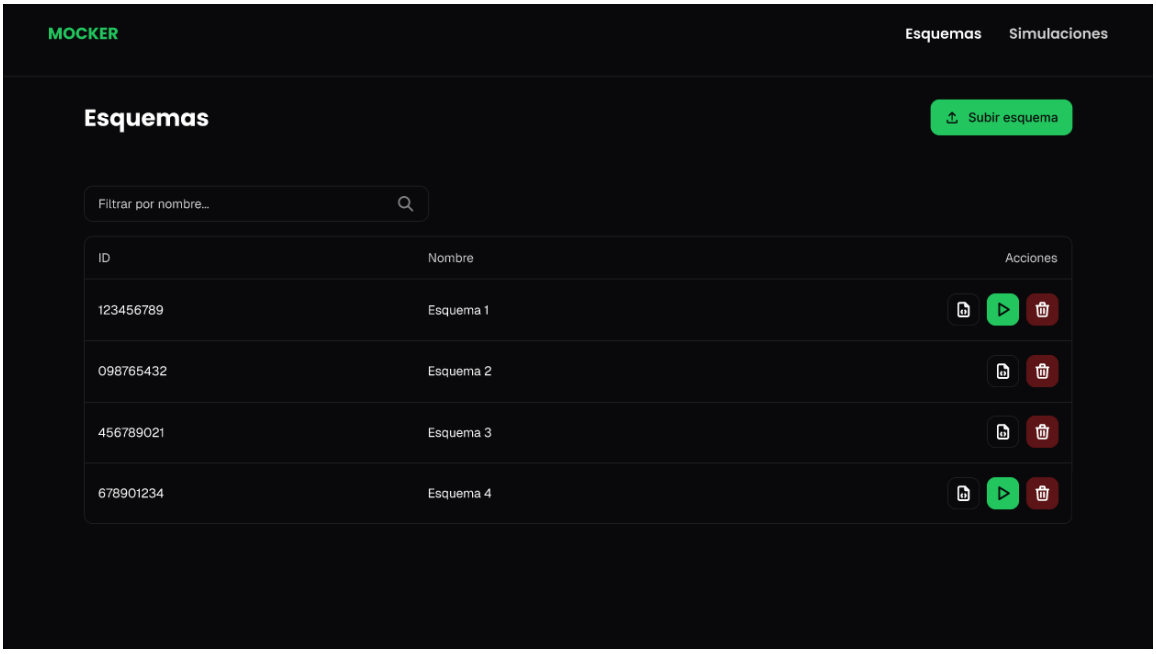


Figura 48

Vista de Simulaciones

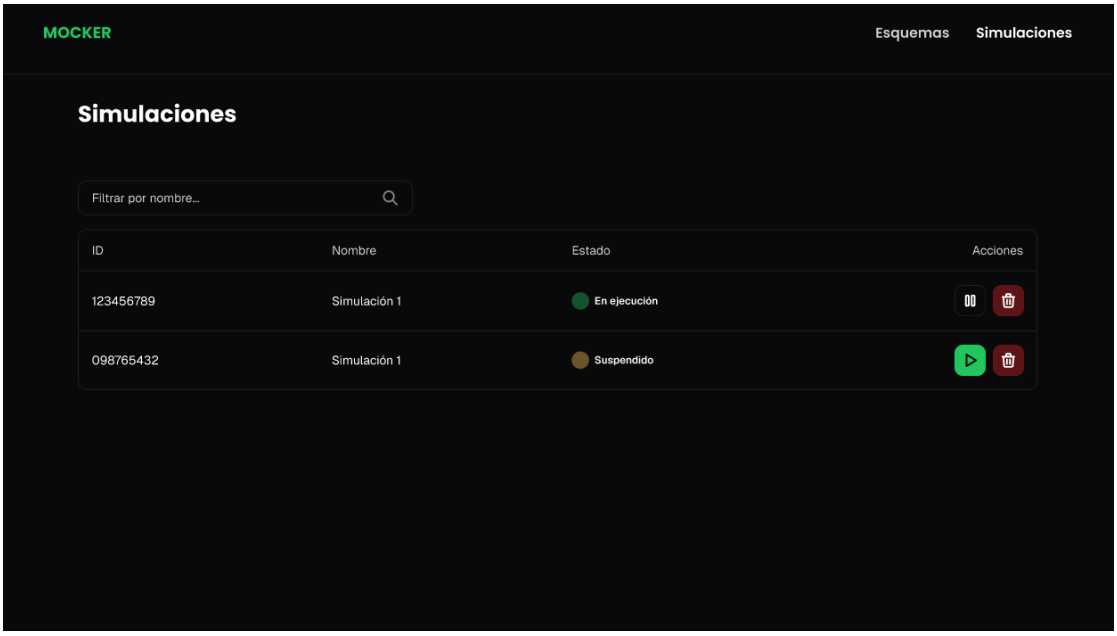


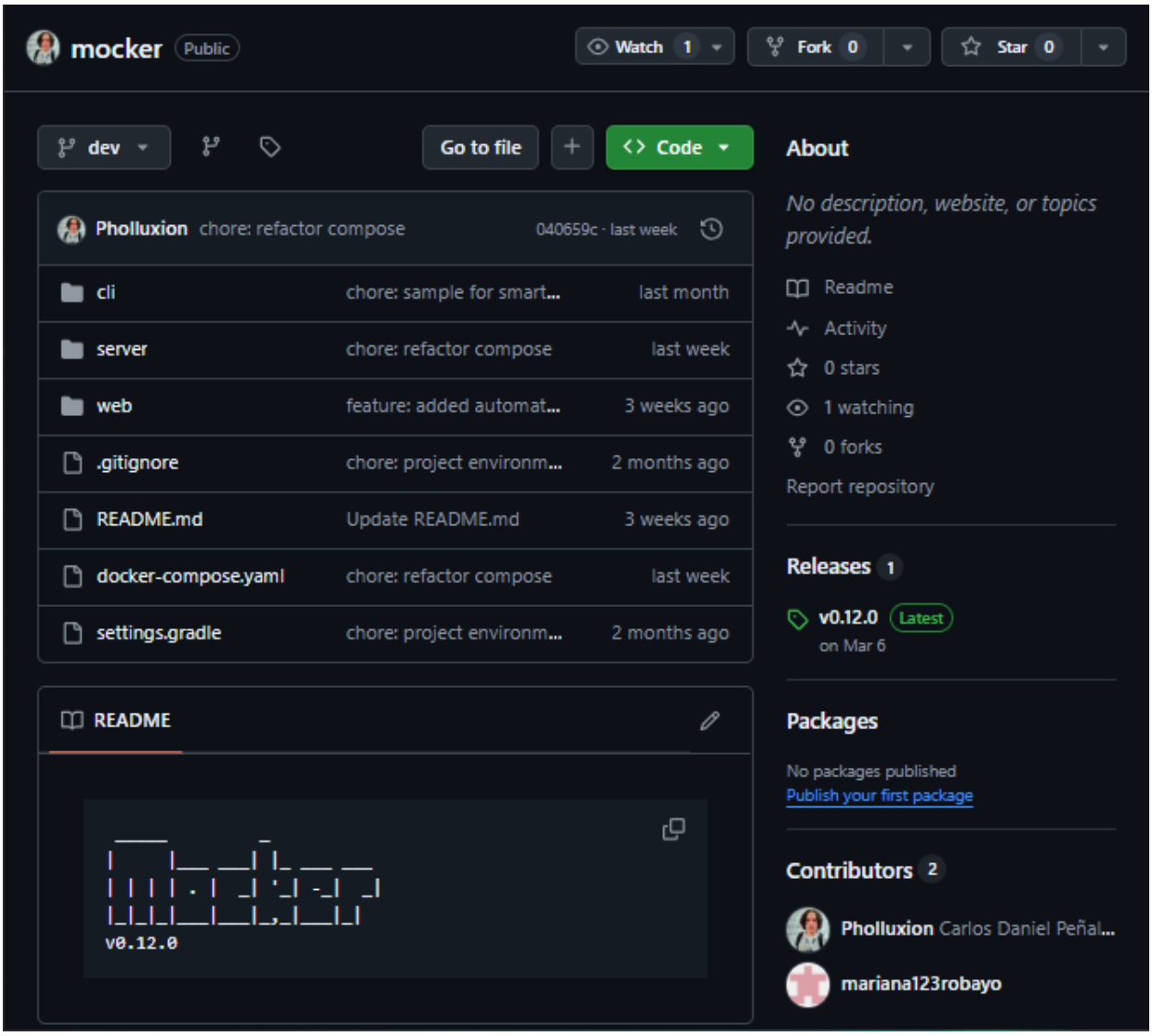
Figura 49

Vista de Logs.

Apéndice I. Repositorio del Proyecto

Figura 55

Repositorio del proyecto



Nota. Tomado de Pholluxion. (2025). GitHub - Pholluxion/mocker.

Este repositorio contiene la interfaz web, CLI y el *backend*.

https://github.com/Pholluxion/mocker

Apéndice J. Ejecución de Pruebas

Durante la etapa de pruebas, se evaluó la funcionalidad y el rendimiento del sistema para garantizar su correcto funcionamiento en distintos escenarios de uso. Se llevaron a cabo pruebas funcionales para validar el comportamiento esperado de cada componente (*backend*, CLI e interfaz web) y pruebas de carga y estrés para medir su desempeño bajo condiciones de alta demanda.

Figura 56

Escenario 1: Simulación Básica.

```yaml
name: Escenario Basico

protocols:
  - type: mqtt
    host: localhost
    port: 1883
    topic: device-messages
    clientId: mocker-client-1
    username: user
    password: password

sampler:
  type: loop
  delay: 1000

generators:

  - type: timestamp
    name: TimestampGenerator
```

Figura 57

Escenario 2: Simulación Intermedia

```yaml
name: Escenario Intermedio

protocols:
  - type: mqtt
    host: localhost
    port: 1883
    topic: device-messages
    clientId: mocker-client-1
    username: user
    password: password

  - type: amqp
    host: localhost
    port: 5672
    username: guest
    password: guest
    exchange: simulation_exchange
    routingKey: simulation.key

sampler:
  type: sequential
  steps:
    - type: step
      duration: 10000
      interval: 1000
    - type: pulse
      pulse: 5000
      idle: 100
    - type: loop
      delay: 3000

generators:
  - type: boolean
    name: BooleanGenerator
    probability: 0.8

  - type: timestamp
    name: TimestampGenerator

  - type: random_integer
    name: RandomInteger
    min: 10
    max: 100

  - type: random_double
    name: RandomDouble
    min: 10
    max: 100
    decimals: 3
```

Figura 58

Escenario 3: Simulación Avanzada

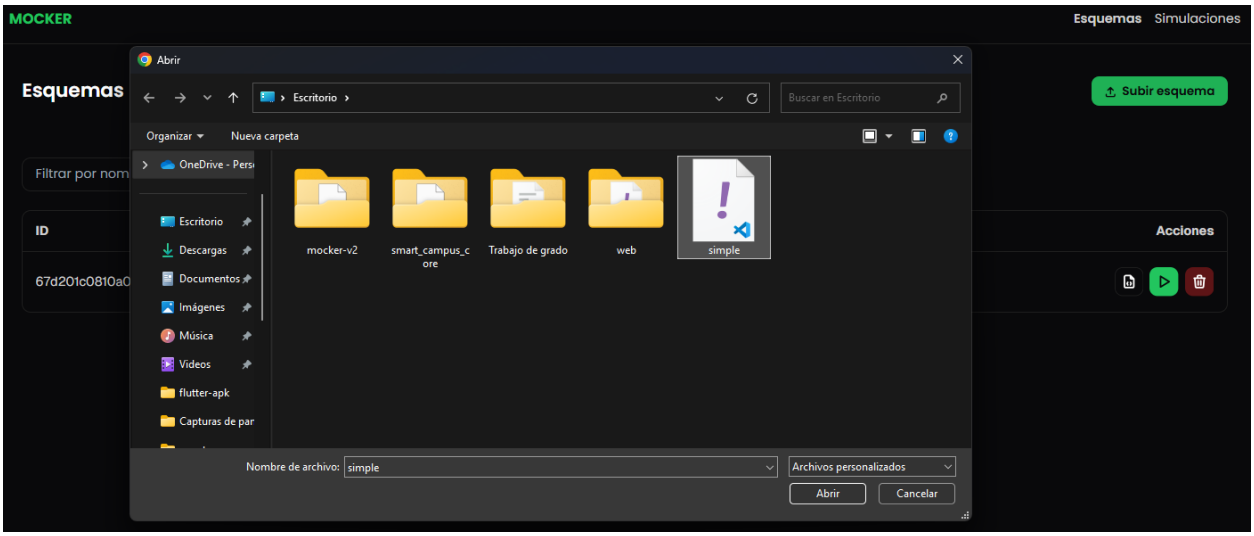
```yaml
name: Escenario Avanzado

protocols:
  - type: mqtt
    host: localhost
    port: 1883
    topic: device-messages
    clientId: mocker-client-1
    username: user
    password: password

  - type: amqp
    host: localhost
    port: 5672
    username: guest
    password: guest
    exchange: simulation_exchange
    routingKey: simulation.key

sampler:
  type: sequential
  steps:
    - type: step
      duration: 10000
      interval: 1000
    - type: delay
      delay: 500
    - type: count
      delay: 200
      count: 20
    - type: random
      min: 100
      max: 1000
    - type: loop
      delay: 1000

generators:
  - type: continuous_log_normal
    name: ContinuousLogNormalGenerator
    mean: 2
    stddev: 0.5
    decimals: 2

  - type: discrete_poisson
    name: DiscretePoissonGenerator
    lambda: 1

  - type: string
    name: StringGenerator
    sampling: random
    values:
      - dev-A
      - dev-B
      - dev-C
```

Interfaz Web

Se verificó que la interfaz web permitiera la configuración y ejecución de simulaciones de manera intuitiva. Se realizaron pruebas de carga de archivos YAML, inicio y monitoreo de simulaciones en tiempo real, así como la correcta finalización de estas. Los resultados confirmaron que la interfaz respondía de manera eficiente a las acciones del usuario y mostraba los datos en tiempo real sin retrasos significativos.

Figura 59

F01 - Carga de configuración



Se verificó que la interfaz web permitiera cargar archivos YAML con la configuración de las simulaciones. Se evaluó que el contenido se mostrara correctamente, sin errores de formato ni pérdidas de información.

Figura 60

F02.1 - Inicio de simulación.

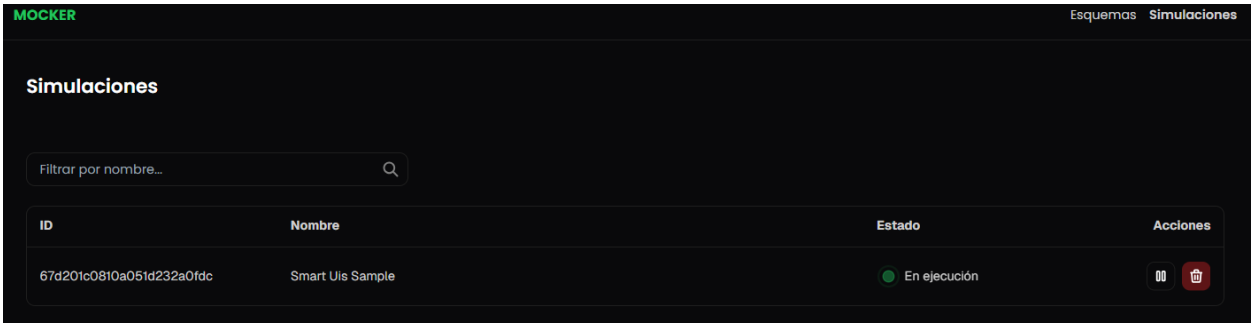
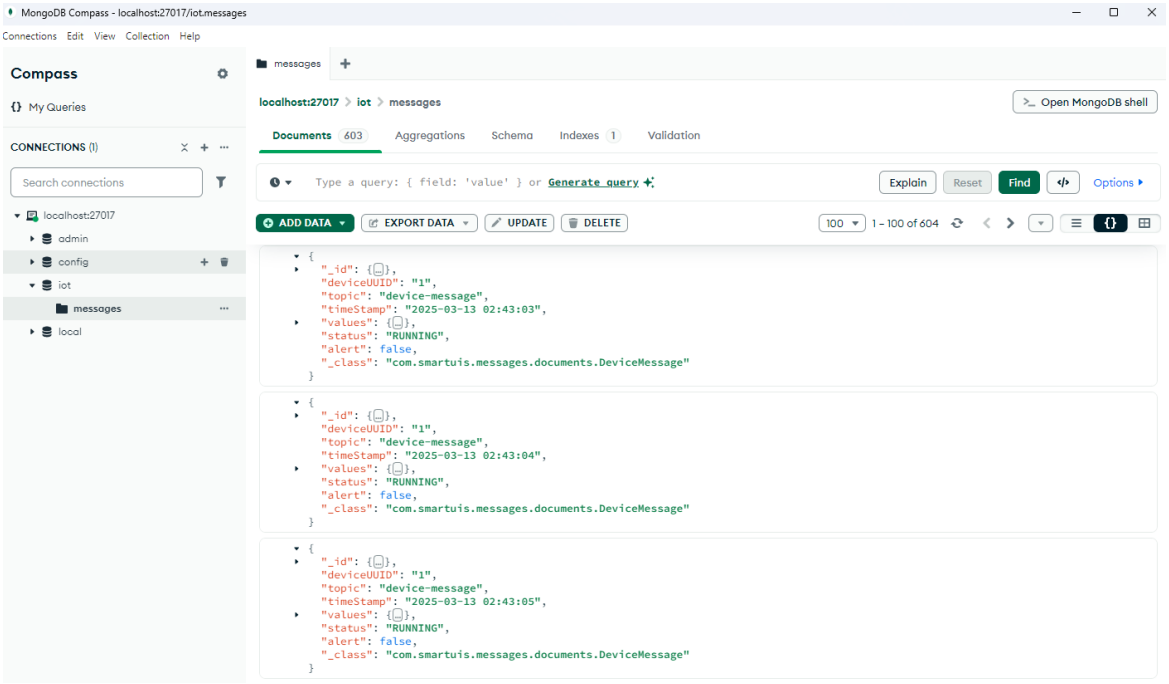


Figura 61

F02.2 – Datos generados



Se comprobó que se pudieran iniciar una simulación desde la interfaz, asegurando que el sistema generara datos según la configuración definida. Se validó que no hubiera fallos en la interpretación del YAML ni errores en la ejecución del muestreo y los generadores.

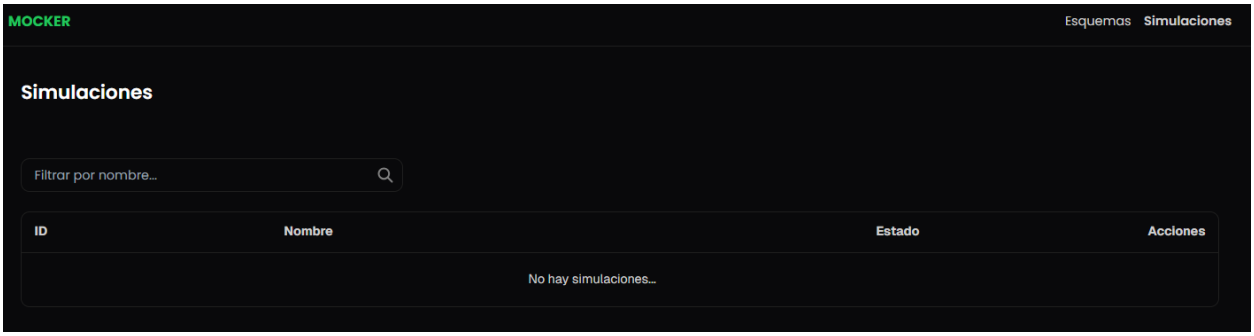
Figura 62

F03 - Monitoreo en tiempo real

Se evaluó que los datos generados por la simulación se reflejaran en tiempo real en la interfaz.

Figura 63

F04 - Finalización de simulación



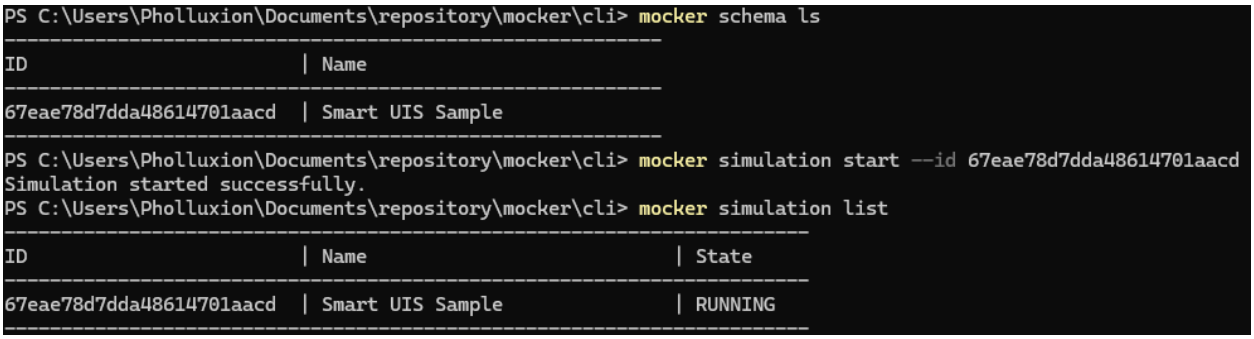
Se probó que se pudiera detener una simulación en ejecución y que el sistema registrara correctamente el estado final. Se revisó que la interfaz reflejara la finalización sin inconsistencias en los datos almacenados.

CLI

Se probó la ejecución de comandos para gestionar simulaciones mediante la CLI. Se validó que se pudieran iniciar y detener simulaciones, así como obtener un listado de las simulaciones activas. Todas las pruebas fueron exitosas, demostrando que la CLI permitía una interacción rápida y eficiente con el sistema.

Figura 64

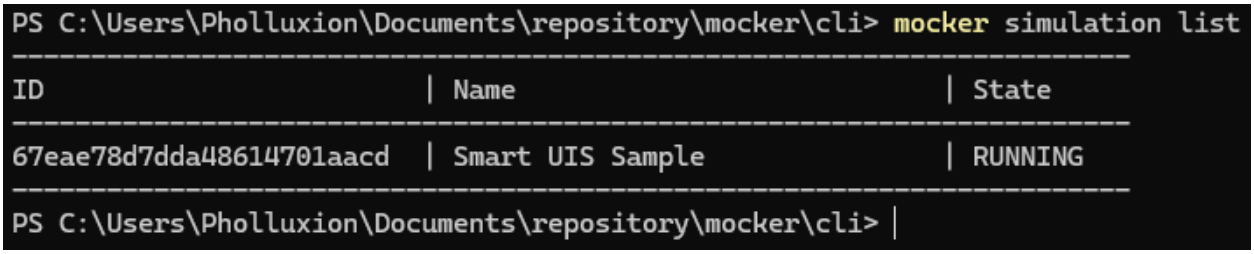
F01 - Ejecución de simulación.



Se validó que los usuarios pudieran iniciar una simulación mediante la CLI, proporcionando los parámetros adecuados. Se verificó que la simulación se ejecutara correctamente y que los datos se generaran conforme a la configuración ingresada.

Figura 65

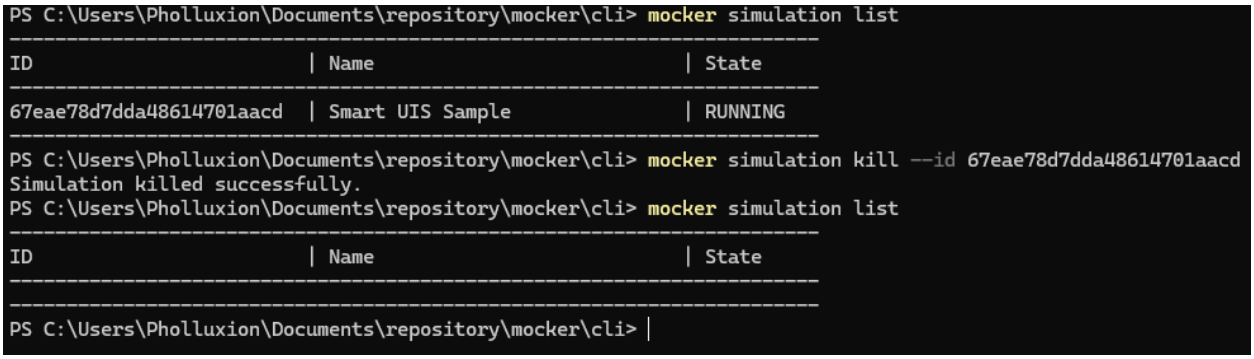
F02 - Listado de simulaciones.



Se probó que la CLI permitiera consultar y listar las simulaciones en ejecución. Se verificó que los detalles mostrados fueran correctos y que los datos de cada simulación coincidieran con los valores esperados.

Figura 66

F03 - Finalización de simulación.



Se evaluó la capacidad de la CLI para detener simulaciones activas. Se comprobó que el comando de finalización funcionara correctamente y que la simulación se cerrara sin afectar otras ejecuciones en el sistema.

Backend

Se evaluó la capacidad del *backend* para procesar archivos YAML, gestionar estados de simulación y almacenar datos en la base de datos. Se comprobó que la máquina de estados basada en Spring State Machine manejara correctamente las transiciones entre los diferentes estados de una simulación. Los resultados indicaron que el *backend* procesó las configuraciones sin errores y garantizó la persistencia de los datos de manera estable.

Figura 67

F01 - Procesamiento de YAML

```
PS C:\Users\Pholluxion\Documents\repository\mocker\cli> curl http://localhost:8090/api/v1/schema


StatusCode        : 200
StatusDescription :
Content           : [{"id":"67eae78d7dda48614701aacd","name":"Smart UIS Sample","sampler":{"type":"loop","delay":1000},
                    "protocols":[{"type":"mqtt","host":"mosquitto","port":1883,"topic":"device-messages","clientId":"mo
                    ck...
RawContent        : HTTP/1.1 200
                    Vary: Origin,Access-Control-Request-Method,Access-Control-Request-Headers
                    Transfer-Encoding: chunked
                    Keep-Alive: timeout=60
                    Connection: keep-alive
                    Content-Type: application/json
                    Da...
Forms             : {}
Headers           : {[Vary, Origin,Access-Control-Request-Method,Access-Control-Request-Headers], [Transfer-Encoding,
                    chunked], [Keep-Alive, timeout=60], [Connection, keep-alive]...}
Images            : {}
InputFields       : {}
Links             : {}
ParsedHtml        : mshtml.HTMLDocumentClass
RawContentLength  : 681
```

Se verificó que el *backend* pudiera recibir y de serializar correctamente los archivos YAML con la configuración de simulación. Se evaluó que los valores se interpretaran adecuadamente y que no hubiera errores de formato o datos faltantes.

Figura 68

F02.1 - Gestión de estados (Created).

```
PS C:\Users\Pholluxion\Documents\repository\mocker\cli> curl http://localhost:8090/api/v1/simulation


StatusCode        : 200
StatusDescription :
Content           : [{"id":"67eae78d7dda48614701aacd","name":"Smart UIS Sample","state":"CREATED"}]
RawContent        : HTTP/1.1 200
                    Vary: Origin,Access-Control-Request-Method,Access-Control-Request-Headers
                    Transfer-Encoding: chunked
                    Content-Type: application/json
                    Date: Mon, 31 Mar 2025 20:02:15 GMT

                    [{"id":"67e...
Forms             : {}
Headers           : {[Vary, Origin,Access-Control-Request-Method,Access-Control-Request-Headers], [Transfer-Encoding,
                    chunked], [Content-Type, application/json], [Date, Mon, 31 Mar 2025 20:02:15 GMT]}
Images            : {}
InputFields       : {}
Links             : {}
ParsedHtml        : mshtml.HTMLDocumentClass
RawContentLength  : 79
```

Figura 69

F02.2 - Gestión de estados (Running).

```
PS C:\Users\Pholluxion\Documents\repository\mocker\cli> curl http://localhost:8090/api/v1/simulation


StatusCode        : 200
StatusDescription :
Content           : [{"id":"67eae78d7dda48614701aacd","name":"Smart UIS Sample","state":"RUNNING"}]
RawContent        : HTTP/1.1 200
                    Vary: Origin,Access-Control-Request-Method,Access-Control-Request-Headers
                    Transfer-Encoding: chunked
                    Keep-Alive: timeout=60
                    Connection: keep-alive
                    Content-Type: application/json
                    Da...
Forms             : {}
Headers           : {[Vary, Origin,Access-Control-Request-Method,Access-Control-Request-Headers], [Transfer-Encoding,
                    chunked], [Keep-Alive, timeout=60], [Connection, keep-alive]...}
Images            : {}
InputFields       : {}
Links             : {}
ParsedHtml        : mshtml.HTMLDocumentClass
RawContentLength  : 79
```

Figura 70

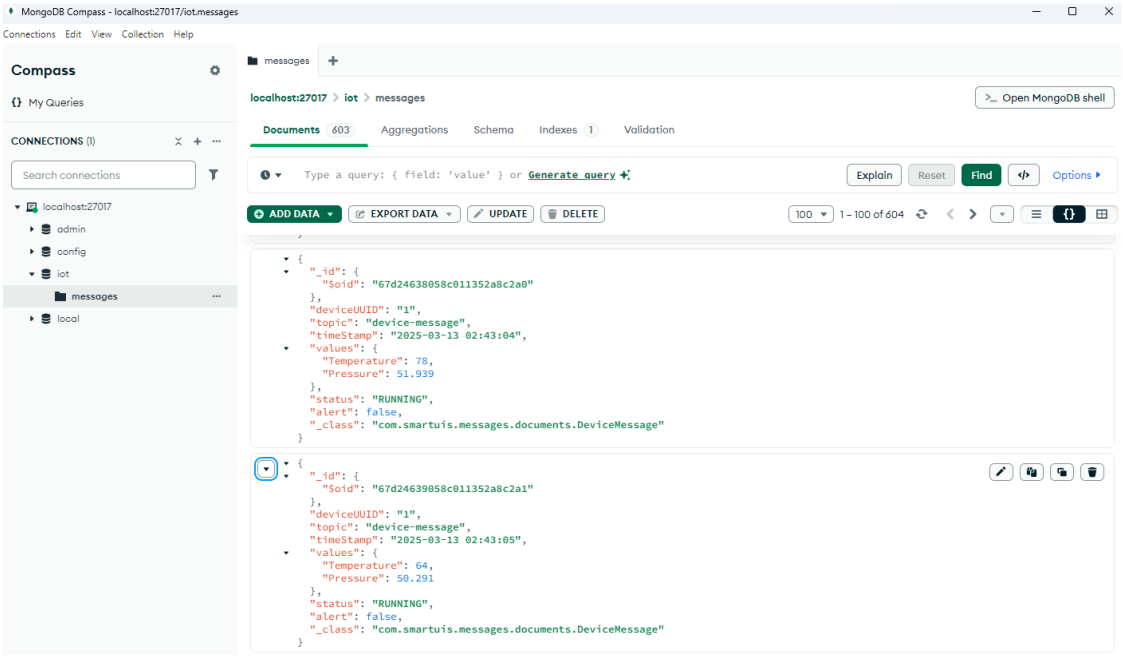
F02.3 - Gestión de estados (Stopped).

```
PS C:\Users\Pholluxion\Documents\repository\mocker\cli> curl http://localhost:8090/api/v1/simulation


StatusCode        : 200
StatusDescription :
Content           : [{"id":"67eae78d7dda48614701aacd","name":"Smart UIS Sample","state":"STOPPED"}]
RawContent        : HTTP/1.1 200
                    Vary: Origin,Access-Control-Request-Method,Access-Control-Request-Headers
                    Transfer-Encoding: chunked
                    Content-Type: application/json
                    Date: Mon, 31 Mar 2025 19:56:07 GMT

                    [{"id":"67e...
Forms             : {}
Headers           : {[Vary, Origin,Access-Control-Request-Method,Access-Control-Request-Headers], [Transfer-Encoding,
                    chunked], [Content-Type, application/json], [Date, Mon, 31 Mar 2025 19:56:07 GMT]}
Images            : {}
InputFields       : {}
Links             : {}
ParsedHtml        : mshtml.HTMLDocumentClass
RawContentLength  : 79
```

Se validó que la máquina de estados basada en Spring State Machine manejara correctamente las transiciones de cada simulación. Se revisó que los cambios de estado ocurrieran en el orden esperado y que no hubiera estados inconsistentes.

Figura 71

F03 - Persistencia de datos.



Se comprobó que el sistema almacenara correctamente las simulaciones finalizadas en la base de datos. Se evaluó que los registros guardados coincidieran con los datos generados y que fueran accesibles para su posterior consulta.

Figura 72

Uso de Memoria y CPU al Inicio con 5 Simulaciones

CONTAINER ID	NAME	CPU %	MEM USAGE
57a23d759942	mocker-web-1	0.00%	9.465MiB
0f30265384ec	mocker-server-1	0.02%	131.2MiB
0e944b18297a	mocker-database-1	0.31%	199.5MiB

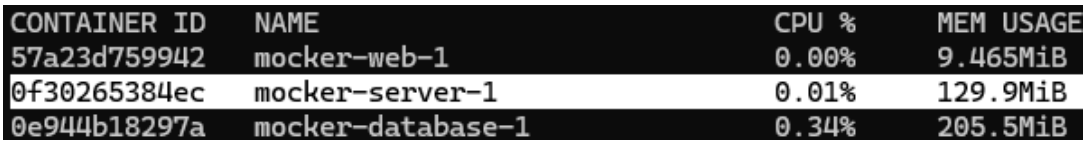
Figura 73

Uso de Memoria y CPU al Final con 5 Simulaciones

CONTAINER ID	NAME	CPU %	MEM USAGE
57a23d759942	mocker-web-1	0.00%	9.465MiB
0f30265384ec	mocker-server-1	0.92%	154.2MiB
0e944b18297a	mocker-database-1	0.33%	205.5MiB

Figura 74

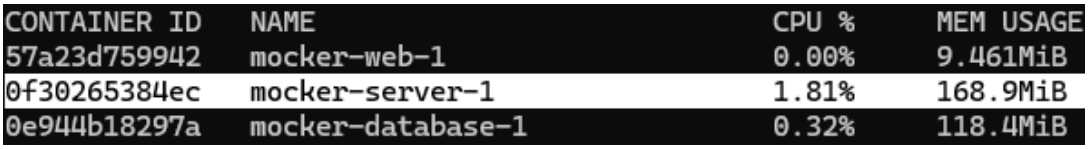
Uso de Memoria y CPU al Inicio con 10 Simulaciones



CONTAINER ID	NAME	CPU %	MEM USAGE
57a23d759942	mocker-web-1	0.00%	9.465MiB
0f30265384ec	mocker-server-1	0.01%	129.9MiB
0e944b18297a	mocker-database-1	0.34%	205.5MiB

Figura 75

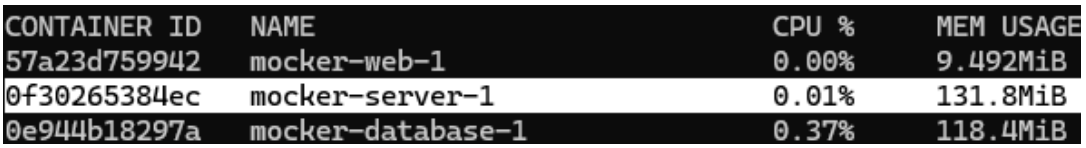
Uso de Memoria y CPU al Final con 10 Simulaciones



CONTAINER ID	NAME	CPU %	MEM USAGE
57a23d759942	mocker-web-1	0.00%	9.461MiB
0f30265384ec	mocker-server-1	1.81%	168.9MiB
0e944b18297a	mocker-database-1	0.32%	118.4MiB

Figura 76

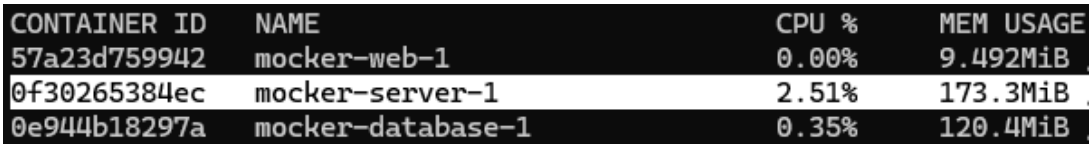
Uso de Memoria y CPU al Inicio con 15 Simulaciones



CONTAINER ID	NAME	CPU %	MEM USAGE
57a23d759942	mocker-web-1	0.00%	9.492MiB
0f30265384ec	mocker-server-1	0.01%	131.8MiB
0e944b18297a	mocker-database-1	0.37%	118.4MiB

Figura 77

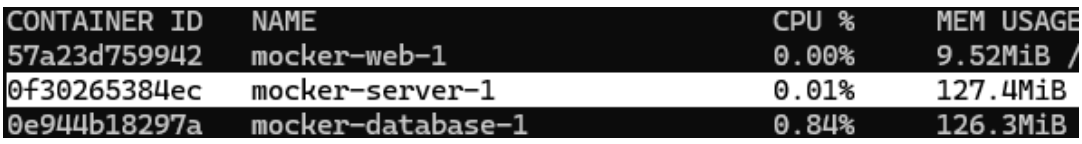
Uso de Memoria y CPU al Final con 15 Simulaciones



CONTAINER ID	NAME	CPU %	MEM USAGE
57a23d759942	mocker-web-1	0.00%	9.492MiB
0f30265384ec	mocker-server-1	2.51%	173.3MiB
0e944b18297a	mocker-database-1	0.35%	120.4MiB

Figura 78

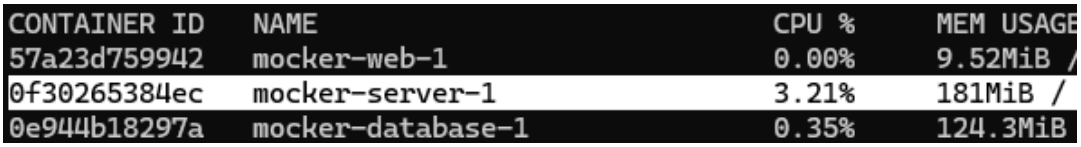
Uso de Memoria y CPU al Inicio con 20 Simulaciones



CONTAINER ID	NAME	CPU %	MEM USAGE
57a23d759942	mocker-web-1	0.00%	9.52MiB /
0f30265384ec	mocker-server-1	0.01%	127.4MiB
0e944b18297a	mocker-database-1	0.84%	126.3MiB

Figura 79

Uso de Memoria y CPU al Final con 20 Simulaciones



CONTAINER ID	NAME	CPU %	MEM USAGE
57a23d759942	mocker-web-1	0.00%	9.52MiB /
0f30265384ec	mocker-server-1	3.21%	181MiB /
0e944b18297a	mocker-database-1	0.35%	124.3MiB