Facultad de Ingeniería
Escuela de Ingeniería Informática

DESARROLLO DE UN ENTORNO WEB PARA LA SIMULACIÓN DE PLATAFORMAS DE PROCESAMIENTO DE STREAMS ORIENTADA A SERVICIOS

Por

Bastián Joel Toledo Salas

Trabajo realizado para optar al Título de
INGENIERO (CIVIL) EN INFORMÁTICA
Prof. Guía: Alonso Inostrosa Psijas
Enero 2025

Certifico que he leído este documento y que, en mi opinión, es adecuado en ámbito y calidad como trabajo para optar al título de Ingeniero Civil en Informática.

Alonso Inostrosa Psijas Profesor Guía

Certifico que he leído este documento y que, en mi opinión, es adecuado en ámbito y calidad como trabajo para optar al título de Ingeniero Civil en Informática.

Nombre Profesor Informante 1 Profesor Informante

Aprobado por la Escuela de Ingeniería en Informática, UNIVERSIDAD DE VALPARAÍSO.

Resumen

El procesamiento de streams es fundamental para gestionar grandes volúmenes de datos en tiempo real, pero las simulaciones tradicionales de plataformas de procesamiento de streams a menudo tienen limitaciones en escalabilidad y flexibilidad. La adopción de arquitecturas de microservicios y contenedores ha demostrado ser una solución efectiva para superar estas barreras.

Este trabajo aborda la necesidad de escalar el simulador SimStream a un entorno web, utilizando una arquitectura de servicios (SOA) y diversas tecnologías de backend y frontend. La solución propuesta encapsula el simulador en contenedores Docker, integrando RabbitMQ para la gestión de colas, MongoDB como base de datos y frameworks como NestJS y NextJS para el desarrollo del backend y frontend, respectivamente.

El resultado principal fue el desarrollo de una plataforma web que permite simular y gestionar modelos de procesamiento de streams de manera eficiente, con una infraestructura escalable y modular. Esta solución mejora la flexibilidad y escalabilidad del simulador, optimizando su rendimiento a través del uso de contenedores y microservicios.

Los resultados obtenidos contribuyen al estado del arte en la simulación de plataformas de procesamiento de streams, facilitando la simulación interactiva de modelos de datos en tiempo real. Además, esta infraestructura abre nuevas posibilidades para el análisis y simulación de sistemas distribuidos, lo que podría tener aplicaciones en diversas disciplinas tecnológicas e industriales.

Finalmente, el enfoque adoptado ofrece una base sólida para futuros desarrollos y mejoras, ampliando la capacidad de simulación y adaptabilidad a nuevos escenarios tecnológicos.

Palabras clave: Procesamiento de streams, Simulación de plataformas, Arquitectura de servicios (SOA), Contenedores Docker, RabbitMQ, MongoDB, NestJS, Simulador SimStream.

Agradecimientos

Deseo expresar mi más sincero agradecimiento a todas las personas e instituciones que han sido fundamentales en la realización de este trabajo de investigación. Sin su apoyo, este proyecto no habría sido posible.

En primer lugar, quiero agradecer profundamente a mi Director de Escuela, *Roberto Muñoz*, por su apoyo incondicional y su orientación experta durante todo el proceso. Su paciencia, su enfoque crítico y sus valiosas recomendaciones fueron determinantes para la correcta orientación de mi trabajo y el desarrollo de las ideas presentadas en esta tesis.

Mi más sincero agradecimiento al Jefe de Carrera, *Rodrigo Olivares Órdenes*, por su respaldo constante y su confianza desde el inicio de este proyecto. A pesar de las dificultades que surgieron durante el proceso, siempre estuvo disponible para ofrecerme su apoyo y guía. Sus ideas y sugerencias contribuyeron de manera significativa a la mejora del trabajo, permitiéndome enriquecer el enfoque y las soluciones planteadas.

Agradezco también a mi profesor guía, *Alonso Inostroza Psijas*, quien me brindó su conocimiento y experiencia para superar los obstáculos conceptuales y técnicos que surgieron durante la investigación. Su intervención fue clave para afinar los detalles y enriquecer el proceso de desarrollo, brindándome una visión más profunda y amplia de los problemas planteados en el trabajo.

Quiero expresar mi más sincero agradecimiento a mis amigos y familiares, especialmente a mis padres, por su apoyo emocional, su comprensión y su motivación constante. Gracias por ser mi fuente de aliento en los momentos más difíciles, por brindarme su confianza y por siempre estar a mi lado durante este proceso tan desafiante.

Agradezco a mis amigos más cercanos, quienes me acompañaron a lo largo de este proceso y me ofrecieron su ayuda en diversas situaciones. Su apoyo constante y sus palabras de aliento fueron esenciales para mantenerme motivado y enfocado en mis objetivos.

Por último, quiero agradecer a la *Universidad de Valparaíso* por el respaldo institucional y los recursos proporcionados, que me permitieron acceder a las tecnologías clave para el desarrollo y simulación de la plataforma presentada en este trabajo.

A todos los que han contribuido directa o indirectamente a la culminación de este proyecto, su apoyo ha sido invaluable. Gracias por hacer posible la finalización de este trabajo y por contribuir al crecimiento personal y académico que ha representado esta investigación.

Índice general

Lista de Figuras

Índice de tablas

Capítulo 1

Introducción

El área de estudio de este trabajo de título se centra en el desarrollo de un entorno web para la simulación de plataformas de procesamiento de streams orientado a servicios [7, 8, 9]. Este campo de investigación se sitúa en la intersección de la informática distribuida, la ciencia de datos y la ingeniería de software, y se enmarca dentro del contexto de la creciente demanda de soluciones tecnológicas capaces de manejar grandes volúmenes de datos en tiempo real. En particular, su aplicación es crucial en áreas como la analítica de datos, la Internet de las Cosas (IoT, por sus siglas en inglés) y la detección de anomalías [9].

Las plataformas de procesamiento de streams son sistemas diseñados para analizar, procesar y responder a flujos continuos de datos en tiempo real. Estas plataformas son fundamentales para aplicaciones que generan grandes cantidades de datos que deben ser procesados en tiempo real, como el monitoreo de sensores, el análisis de tráfico de red [10], la detección de amenazas de seguridad y la analítica de datos en tiempo real. Un ejemplo prominente de una plataforma de procesamiento de streams es *Apache Kafka* [11], que proporciona una infraestructura distribuida para el almacenamiento y procesamiento de datos de manera escalable [12]. Otras herramientas populares incluyen *Apache Flink* [13], que ofrece capacidades avanzadas de procesamiento de streams y procesamiento de datos por lotes en un solo motor unificado [14], y *Spark Streaming* [15], que forma parte del ecosistema de *Apache Spark* y permite el procesamiento de datos de manera escalable [5].

Las plataformas de procesamiento de streams enfrentan varios desafíos y problemas que deben ser abordados para garantizar su eficacia y utilidad en entornos del mundo real. Entre estos desafíos destaca la necesidad de procesar grandes cantidades de datos en segundos, con latencias extremadamente bajas, para proporcionar información en tiempo real a medida que los datos arriban [3]. Los componentes de la aplicación y la infraestructura deben garantizar un alto nivel de tolerancia a fallos, evitando la pérdida de datos en caso de fallos en el sistema. Además, la complejidad en la configuración y uso de estas aplicaciones, junto con los obstáculos que la portabilidad y el despliegue representan tanto para usuarios experimentados como para novatos, requieren soluciones adecuadas.

1.1. Principales Contribuciones

Para abordar los desafíos identificados en este trabajo de título, se propone diseñar, desarrollar e implementar un entorno web basado en una arquitectura de servicios y el uso de contenedores Docker [16] como elementos clave. Esta propuesta responde a la necesidad de enfrentar las complejidades asociadas con la creación de modelos concurrentes o paralelos en plataformas de procesamiento de datos. El objetivo principal es proporcionar una herramienta versátil y accesible para la experimentación, modelado, simulación y análisis de resultados, simplificando el proceso de construcción y simulación de modelos para plataformas de procesamiento de streams.

Para lograr este objetivo, se emplea la metodología ágil Scrum [17], que facilita la entrega incremental de funcionalidades y ofrece mayor flexibilidad para adaptarse a los cambios en los requisitos del proyecto a lo largo del tiempo.

Entre las contribuciones esperadas se destacan: el desarrollo de un entorno web para la simulación de plataformas de procesamiento de streams orientado a servicios; la evaluación de la efectividad y el rendimiento del entorno desarrollado en comparación con otras herramientas similares; la documentación exhaustiva del proceso de desarrollo, que incluirá el diseño, implementación y pruebas del sistema; así como la identificación de áreas de mejora y la propuesta de futuras líneas de investigación en este campo.

1.2. Estructura del Documento

El presente documento de tesis se organiza en los siguientes capítulos: En el Capítulo 2, se establece el marco conceptual para entender el contexto de estudio, con definiciones de conceptos clave, junto con una revisión del estado del arte y una discusión sobre trabajos relacionados. En el Capítulo 3, se realiza un análisis detallado de los problemas actuales en las plataformas de procesamiento de streams, se propone una solución a dichos desafíos y se establecen los objetivos del proyecto. En el Capítulo 4, se describe el diseño de la solución propuesta, complementado con diagramas arquitectónicos. En el Capítulo 5, se detallan los recursos de hardware y software utilizados, los lenguajes de programación seleccionados y las estrategias de implementación. En el Capítulo 6, se presentan los resultados obtenidos de las pruebas realizadas sobre el entorno web, incluyendo análisis de casos de uso específicos y validación de los objetivos alcanzados. El capítulo 7 se describe el plan de implantación propuesto para el entorno web desarrollado, considerando los pasos necesarios para llevarlo a un entorno real de producción. Finalmente, en el Capítulo 8, se abordan las conclusiones y se proponen futuras líneas de investigación sobre la simulación de plataformas de procesamiento de streams.

Capítulo 2

Marco Conceptual y Estado del Arte

2.1. Marco Conceptual

Esta sección constituye el fundamento teórico necesario para comprender en su totalidad la temática central abordada en este trabajo de tesis. En ella se presentan los conceptos clave y definiciones relacionadas con el procesamiento y la simulación de plataformas de procesamiento de streams, estableciendo así las bases para el análisis y el diseño de la solución propuesta.

2.1.1. Modelos de Datos en Sistemas de Procesamiento de Streams

Modelo Relacional de Streaming

En los primeros sistemas de procesamiento de streams, se implementó el modelo relacional de streaming, representado por plataformas como Borealis [18], Aurora [19], Stream [20], CDER [21], TelegraphCQ [22] y Gigascope [23].

En este modelo, un stream es interpretado como la descripción de una relación cambiante sobre un esquema común. Además, se asume que los elementos de un stream están acompañados de marcas de tiempo (*timestamps*) o números de secuencia, lo que permite definir ventanas sobre estos streams. Los streams de datos pueden ser generados por fuentes externas o producidos por consultas continuas. Los operadores, a su vez, generan streams de eventos que describen la vista cambiante calculada sobre el stream de entrada, de acuerdo con la semántica relacional del operador. Tanto la semántica como el esquema de relaciones son definidos e impuestos por el sistema.

Modelo de Streaming de Flujo de Datos

En la segunda generación de sistemas de procesamiento de streams se implementó el modelo de streaming de flujo de datos, representado por plataformas como Google Data-Flow Model [24], Apache Flink [14] y Spark Streaming [7].

Este modelo de streaming se representa mediante un grafo de flujo de datos, es decir, un grafo dirigido G = (E,V) , donde los vértices V corresponde a los operadores y las aristas E representan los streams de datos [25]. Las aplicaciones pueden operar en uno de los tres siguientes modos:

Tiempo de Evento o Aplicación: El momento (tiempo) en que los eventos son generados por las fuentes de datos.

Tiempo de Asimilación o Ingestión: El momento (tiempo) en que los eventos llegan al sistema.

Tiempo de procesamiento: El momento (tiempo) en que los eventos son procesados por los sistemas de streaming.

En los sistemas modernos de flujo de datos de streaming, se puede asimilar cualquier tipo de stream de entrada, independientemente de si sus elementos representan adiciones, eliminaciones o reemplazos.

2.1.2. Métodos de Procesamiento de datos

El procesamiento de datos se divide en 3 enfoques principales: procesamiento por lotes, procesamiento por micro-lotes y procesamiento de streams.

Procesamiento por Lotes

el procesamiento por lotes (*batch*) se define como el análisis de grandes conjuntos de datos estáticos, recopilados durante períodos de tiempo anteriores. Sin embargo, esta técnica tiene una alta latencia, con tiempos de respuesta superiores a 30 segundos. Esto puede resultar inadecuado para aplicaciones que requieren procesamiento en tiempo real, con respuestas en el orden de microsegundos [26]. No obstante, este enfoque puede realizarse de manera casi en tiempo real mediante el procesamiento por micro-lotes.

Procesamiento Micro-Lotes

El procesamiento por micro-lotes (*micro-batch*) trata los streams como una secuencia de pequeños bloques de datos. Para intervalos de tiempo cortos, las entradas que son recibidas se agrupan en bloques de datos y se entregan al sistema para su procesamiento por lotes [3].

Procesamiento de Streams

El procesamiento de streams se ocupa de analizar secuencias masivas de datos ilimitados que se generan de forma continúa. Este enfoque presenta una menor la latencia en comparación con los micro-lotes, ya que los mensajes se procesan inmediatamente después de su llegada. Esto se traduce en un rendimiento superior en tiempo real, aunque con una mayor complejidad en la tolerancia a fallos, ya que debe manejarse por cada mensaje procesado [3].

2.1.3. Procesamiento de Streams

El procesamiento de streams (*Stream Processing*) se define como un método de computación distribuida diseñado para soportar la recopilación y el análisis de grandes volúmenes de datos provenientes de un flujo continuo(*data stream*). Su objetivo principal es facilitar la toma de decisiones en tiempo real [27].

Este enfoque abarca varios conceptos y principios fundamentales que son esenciales para entender su funcionamiento y aplicaciones. A continuación, se describen los conceptos clave:

- Flujo de datos (*Data Stream*): Un Conjunto de datos que se genera incrementalmente a lo largo del tiempo, en lugar de estar disponible en su totalidad antes de comenzar su procesamiento [25]. Estos flujos pueden ser potencialmente ilimitados y reflejan eventos en tiempo real.

- Consulta en tiempo real (*Streaming Query*): Consiste en capturar eventos y producir resultados de forma continua, utilizando un único recorrido o un número limitado de recorridos sobre los datos [25].

- Procesamiento en Tiempo Real (*Real-Time Processing*): Se refiere a la capacidad de analizar y actuar sobre los datos a medida que llegan, con una latencia mínima, asegurando respuestas rápidas sin demoras significativas.

2.1.4. Plataformas de Procesamiento de Streams

Las plataformas de procesamiento de streams han emergido como herramientas esenciales para el análisis de datos en tiempo real. Estas plataformas permiten procesar flujos de datos de manera continua, siendo fundamentales en escenarios donde la velocidad y la precisión son cruciales.

Autores como **Carbone, Paris, et. al.** destacan que estas plataformas están diseñadas para ofrecer latencias extremadamente bajas, asegurando que los resultados del análisis estén disponibles en tiempo real o casi tiempo real.Por otro lado,**Zaharia, M, et. al.** [7]

subrayan que el procesamiento de datos en tiempo real es esencial para una amplia gama de aplicaciones, como redes sociales, publicidad en línea, detección de fraudes y monitorización de infraestructuras críticas.

A diferencia de los sistemas tradicionales de procesamiento por lotes, estas plataformas permiten el procesamiento continuo de datos a medida que llegan, en lugar de esperar la acumulación de un conjunto completo de datos antes de iniciar el análisis y procesamiento. Según **Kreps, Jay**, este modelo representa un cambio de paradigma en el tratamiento de datos, considerándolos como eventos temporales que deben procesarse de forma continua [28].

2.1.5. Clasificación de Plataformas de Procesamiento de Streams

La computación distribuida y paralela han surgido como un solución calve para el procesamiento de grandes conjuntos de datos (*datasets*). Sin embargo, su complejidad y ciertas características pueden limitar su aprovechamiento por usuarios comunes. Entre las principales preocupaciones se incluyen la partición y distribución de datos, escalabilidad, equilibrio de carga, tolerancia a fallos y alta disponibilidad [1].

Para abstraer estas funciones y ofrecer soluciones de alto nivel, se han desarrollado diversas plataformas clasificadas según su enfoque de datos:

- Procesamiento por lotes (*Batch Processing*) : Procesamiento de grandes volúmenes de datos almacenados de manera estática 2.1.2.
- Procesamiento de Streams (*Stream Processing*): Procesamiento continuo de datos en tiempo real 2.1.2.
- Procesamiento Híbrido (*Hybrid Processing*): Combina diferentes técnicas, incluyendo procesamiento por lotes, micro-lotes 2.1.2 y procesamiento de streams en tiempo real.

A continuación, se presenta una figura que ilustra esta clasificación:

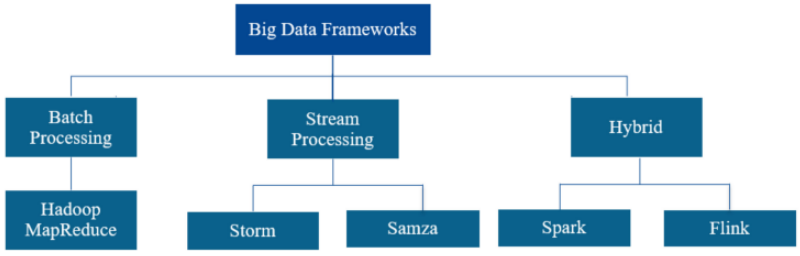

Figura 2.1: Clasificación de Plataformas de Procesamiento. Recuperado de [1]

2.1.6. Modelos de Simulación en Sistemas de Procesamiento de Streams

Los sistemas de procesamiento de streams se benefician enormemente de la simulación para evaluar su rendimiento y eficiencia en diferentes escenarios. La simulación permite a los investigadores modelar el comportamiento de sistemas complejos, realizar experimentos controlados y probar configuraciones antes de implementarlas en entornos reales. A continuación, se describen algunos modelos clave de simulación en este ámbito:

Simulación Basada en Eventos Discretos

La simulación basada en eventos discretos (*Discrete Event Simulation, DES*) es un enfoque ampliamente utilizado para modelar sistemas dinámicos. Este método se centra en los eventos que ocurren a lo largo del tiempo, permitiendo capturar el comportamiento del sistema de manera precisa.

- Evento: Representa un suceso que ocurre en un momento específico dentro del sistema. En el contexto de sistemas computacionales, los eventos pueden incluir la llegada de datos, la ejecución de una tarea, o la generación de resultados [29].
- Procesamiento de Eventos: Operaciones realizadas sobre eventos, como creación, transformación, o eliminación, que determinan el flujo de datos dentro del sistema [29].
- Stream de Eventos: Secuencia ordenada de eventos con marcas de tiempo (*timestamps*), representando el flujo continuo de datos procesados.
- Procesamiento de Eventos Discretos: Técnica que permite procesar y correlacionar grandes volúmenes de datos en tiempo real, facilitando la detección y respuesta a eventos críticos [30].

CEPSim: Modelado y simulación de sistemas de Procesamiento de Eventos Discretos en entornos de nube

CEPSim (*Complex Event Processing Simulator*) fue desarrollado por **Wilson, A. et al.** como una herramienta para modelar y simular sistemas de procesamiento de eventos complejos (*Complex Event Processing, CEP*) en entornos de nube [31]. El objetivo principal de CEPSim es ayudar a investigadores y desarrolladores a entender cómo escalan los sistemas CEP y cómo varían sus desempeños bajo diferentes condiciones.

Características Principales:

- Basado en CloudSim: Utiliza CloudSim, un marco de simulación extensible diseñado para evaluar entornos de computación en la nube. Esto proporciona una base sólida y personalizable para incluir características de procesamiento de eventos complejos.
- Estrategias de Procesamiento de Consultas: Permite analizar cómo diferentes estrategias de procesamiento afectan la latencia, el uso de recursos y el rendimiento general del sistema.
- Modelado de Topologías CEP: Facilita la creación de grafos de procesamiento de consultas, donde los nodos representan operadores de consulta y las aristas representan streams de datos.
- Entorno Controlado: Proporciona la posibilidad de realizar experimentos sin el costo y la complejidad de desplegar sistemas reales, lo que resulta ideal para la investigación y el desarrollo temprano.

Aplicaciones:

- Evaluar el impacto de cambios en la infraestructura, como aumentar o reducir la capacidad de los nodos en la nube.
- Analizar el rendimiento de sistemas CEP bajo diferentes volúmenes de carga.
- Probar nuevas estrategias de asignación de recursos y distribución de datos.

Limitaciones:

- Enfocado únicamente en entornos de nube, por lo que no considera arquitecturas híbridas o sistemas Edge-Cloud.
- Requiere conocimientos avanzados para modelar las topologías de consultas adecuadamente.

ECSSim: Un simulador para dispositivos de borde en la nube

ECSSim es otra contribución de **Amarasinghe, K. et al.**, orientada específicamente al modelado y simulación de sistemas Edge-Cloud [32]. A diferencia de ECSNeT++, este simulador se centra más en el rendimiento del sistema y la interacción entre dispositivos en el borde y servidores en la nube.

Características Principales:

- Optimización para Sistemas de Borde: Diseñado para modelar escenarios donde los datos se procesan principalmente cerca de los dispositivos finales.
- Evaluación de Desempeño: Permite analizar métricas como la latencia, el uso de ancho de banda y la eficiencia energética de los nodos del borde y la nube.
- Modelado de Interacciones: Proporciona herramientas para estudiar cómo los dispositivos en el borde interactúan con la nube y entre sí.
- Soporte para Recursos Heterogéneos: Considera la variabilidad en las capacidades de procesamiento y almacenamiento de los nodos en el borde.

Aplicaciones:

- Estudio de sistemas IoT con procesamiento en tiempo real.
- Optimización del balance de carga entre dispositivos de borde y servidores en la nube.
- Análisis del impacto de fallos de red en sistemas distribuidos.

Limitaciones:

- Limitado a entornos Edge-Cloud, sin soporte para simulaciones de arquitecturas exclusivamente basadas en la nube.
- Requiere ajustes significativos para adaptarse a escenarios personalizados.

ECSNet++: Un kit de herramientas de simulación para plataformas de procesamiento de streams en la nube.

ECSNeT++ es un simulador desarrollado por **Amarasinghe, K. et al.**, diseñado específicamente para evaluar el rendimiento de plataformas de procesamiento de streams en entornos Edge-Cloud [33]. Este simulador se basa en OMNeT++, un marco ampliamente utilizado para simulaciones de redes y sistemas distribuidos.

Características Principales:

- Plataforma Basada en OMNeT++: Aprovecha el soporte extensivo de OMNeT++ para modelar redes complejas, incluyendo comunicaciones entre nodos en el borde y la nube.
- Procesamiento de Streams: Facilita el modelado de flujos de datos continuos, representando operaciones comunes como filtrado, agregación y análisis en tiempo real.
- Compatibilidad Edge-Cloud: Diseñado para simular sistemas donde los datos se procesan tanto en nodos del borde como en la nube, lo que permite estudiar arquitecturas híbridas.
- Escenarios Personalizables: Permite a los investigadores explorar configuraciones como el balance de carga entre nodos, la asignación de tareas y el impacto de la latencia en el rendimiento.

Aplicaciones:

- Simulación de arquitecturas Edge-Cloud para servicios sensibles a la latencia, como IoT y sistemas autónomos.
- Evaluación de estrategias de asignación de recursos en entornos distribuidos.
- Estudio del impacto de redes de baja calidad o congestionadas en el procesamiento de datos.

Limitaciones:

- Complejidad en la configuración inicial debido a la necesidad de comprender OMNeT++ y su ecosistema.
- Dependencia de arquitecturas Edge-Cloud, lo que puede no ser ideal para sistemas puramente centralizados o descentralizados.

Evaluación de rendimiento basada en simulación de eventos discretos de sistemas de procesamiento de streams

En la investigación presentada en la Conferencia Internacional IEEE, los autores **Muthukrishnan, P. y Fadnis, K** [34] proponen un enfoque novedoso para evaluar el rendimiento de sistemas de procesamiento de streams mediante la simulación de eventos discretos o DES (*Discrete Event Simulation*). Este enfoque consiste en modelar el comportamiento de un sistema de procesamiento de streams y simular su funcionamiento bajo distintas condiciones de carga de trabajo. La metodología permite evaluar aspectos clave como la eficiencia, la latencia y la escalabilidad, entre otras métricas de desempeño.

Características Principales del Enfoque:

- Modelado Preciso de Procesos: El uso de DES permite capturar la dinámica del sistema, representando eventos como llegadas de datos, operaciones de procesamiento, tiempos de espera en colas y transmisión de resultados.
- Simulación de Cargas Variables: Se pueden generar diferentes escenarios para evaluar cómo responde el sistema a volúmenes de datos crecientes, picos de demanda y cambios en los patrones de entrada.
- Comparación de Configuraciones y Algoritmos: Este enfoque permite a los diseñadores evaluar el impacto de distintas configuraciones de hardware, algoritmos de procesamiento, y estrategias de asignación de recursos.

Aplicaciones:

- Diseño y optimización de plataformas de procesamiento de streams en tiempo real.
- Evaluación de arquitecturas Edge-Cloud y distribuidas para sistemas de IoT o análisis de datos.
- Comparación de algoritmos de procesamiento paralelos o distribuidos.

Limitaciones:

- Dependencia del Modelo: La calidad de los resultados depende directamente de la precisión con que se modelen los eventos y procesos.
- Costos Computacionales: Aunque es más eficiente que desplegar un sistema real, la simulación de grandes cantidades de eventos puede ser computacionalmente costosa.
- Falta de Interacción con el Entorno Real: No permite capturar ciertas dinámicas imprevistas que solo ocurren en entornos reales.

Contribuciones Relevantes:

- La implementación de este enfoque basado en DES ofrece una herramienta poderosa para investigadores, desarrolladores y gestores interesados en comprender y optimizar el rendimiento de sistemas de procesamiento de streams.
- Abre nuevas oportunidades para explorar cómo las variaciones en el diseño o la infraestructura impactan la eficiencia de los sistemas en diferentes escenarios.

2.2. Estado del Arte

2.2.1. Principales Plataformas de Procesamiento de Streams

En esta subsección exploramos algunas de las principales plataformas de procesamiento de streams disponibles en el mercado. Estas plataformas han sido diseñadas para gestionar grandes volúmenes de datos en tiempo real, permitiendo a las organizaciones aprovechar sus flujos de datos de manera eficiente y ágil. A continuación, se destacan las características principales de algunas de ellas:

- Apache Samza [35]: Ofrece un sistema confiable para el procesamiento de datos en tiempo real, con una API intuitiva basada en Apache Kafka Streams. Su diseño está enfocado en la fiabilidad, recuperación ante fallos y alto rendimiento.
- Apache Flink [13]: Se caracteriza por ser una plataforma unificada que combina el procesamiento de datos en tiempo real y el procesamiento por lotes, ofreciendo una arquitectura versátil y escalable.
- Apache Spark Streaming [15]: Con capacidades avanzadas de procesamiento y análisis, permite integrar flujos de datos en tiempo real con tareas analíticas más complejas, siendo una opción popular en entornos distribuidos.
- Apache Storm [36]: Es una plataforma diseñada para aplicaciones que requieren tolerancia a fallos y escalabilidad horizontal. Proporciona un modelo de programación flexible basado en la composición de ”bolts” y ”spouts”.

Estas plataformas están transformando la manera en que las organizaciones gestionan sus datos en tiempo real, abriendo nuevas posibilidades para la innovación, la optimización de procesos y la toma de decisiones ágil.

2.2.2. Apache Samza

Apache Samza [37] es una plataforma diseñada específicamente para el procesamiento con estado de streams de datos en tiempo real y por lotes. Su arquitectura permite manejar grandes volúmenes de datos (*throughput*) manteniendo una alta fiabilidad y tolerancia a fallos. Este sistema ha sido adoptado por grandes compañías como LinkedIn, VMWare, Uber, Netflix y TripAdvisor [35], consolidándose como una solución clave para aplicaciones de misión crítica.

Características Clave de Samza:

- Procesamiento Escalable: Aprovecha un modelo descentralizado sin un maestro global para coordinar actividades, lo que mejora su escalabilidad y resiliencia.

- Gestión de Estado Local: Implementa abstracciones como la captura de registros de cambio (*changelog*) y particiones de streams, permitiendo una recuperación eficiente ante fallos.
- Integración Nativa con Apache Kafka y YARN: Soporta de manera nativa el streaming de datos desde Kafka y la ejecución de tareas distribuidas mediante YARN.

Arquitectura Samza

La arquitectura de Apache Samza se divide en tres capas principales, cada una con responsabilidades específicas para garantizar un procesamiento eficiente de datos (Figura 2.2):

- Capa de Streaming (*Streaming Layer*): Se encarga de proporcionar una fuente reproducible de datos en tiempo real, utilizando sistemas como Apache Kafka [11], AWS Kinesis [38] o Azure EventHub [39].
- Capa de Ejecución (*Execution Layer*): Gestiona los recursos necesarios para la ejecución de tareas, como la planificación y el balanceo de cargas, utilizando herramientas como YARN [40] o Mesos [41].
- Capa de Procesamiento (*Processing Layer*): Es responsable del procesamiento de los datos, gestionando el flujo entre las tareas asignadas.

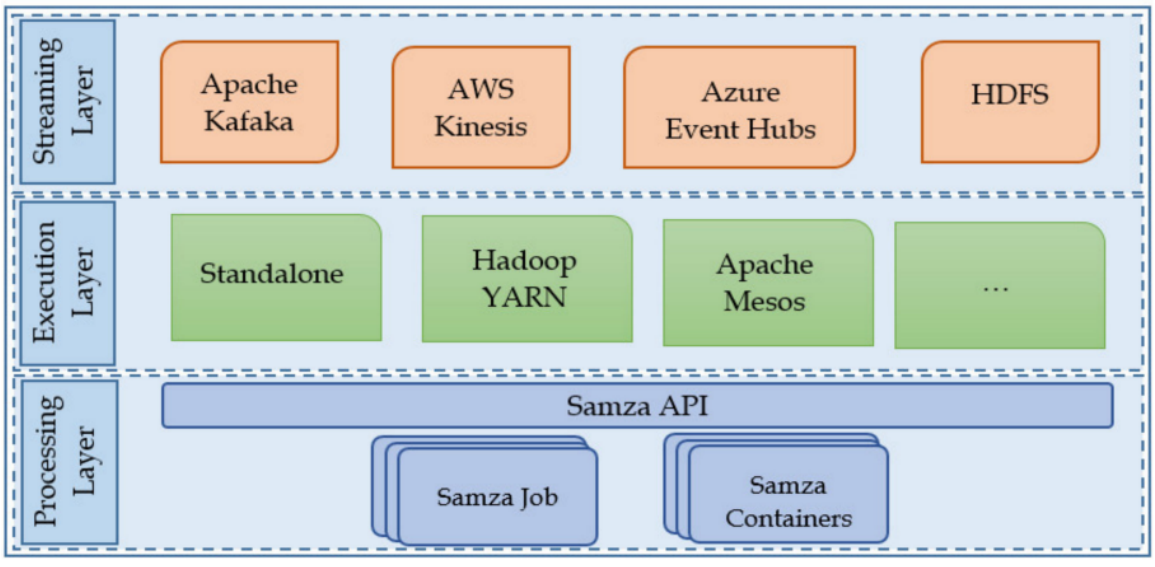

Figura 2.2: Arquitectura y componentes de la plataforma Apache Samza. Recuperado de [1]

Modelo de Ejecución El modelo de ejecución de Apache Samza se basa en un paradigma de publicación y suscripción (*publish/subscribe*), donde las tareas consumen datos de un tópico en Apache Kafka, procesan cada mensaje y envían los resultados a otro stream para un procesamiento adicional. Cada tarea trabaja con particiones de datos basadas en pares clave-valor, lo que permite un procesamiento paralelo eficiente.

- Tareas y Contenedores: Las tareas se asignan a contenedores YARN, que se distribuyen entre los nodos del clúster. Esto asegura un uso equilibrado de los recursos.
- Procesamiento Paralelo: Samza permite la ejecución de múltiples tareas simultáneamente, maximizando el rendimiento al procesar particiones de datos en paralelo.

Fortalezas de Apache Samza

- Alta Resiliencia: Su modelo descentralizado y la captura de registros de cambio garantizan una rápida recuperación tras fallos.
- Escalabilidad Horizontal: Puede manejar fácilmente incrementos en la carga de trabajo al distribuir tareas en nuevos nodos del clúster.
- Integración Flexible: Su compatibilidad con múltiples sistemas de streaming y herramientas de orquestación lo convierten en una opción versátil para diversas aplicaciones.

Limitaciones

- Curva de Aprendizaje: Aunque poderosa, la configuración inicial puede ser compleja para desarrolladores sin experiencia en herramientas de procesamiento distribuido.
- Dependencia de Infraestructura: Requiere un clúster de YARN o Mesos bien configurado, lo que puede incrementar los costos de implementación

Apache Samza ha demostrado ser una solución robusta y eficiente para organizaciones que necesitan manejar flujos de datos de gran volumen en tiempo real. Su arquitectura bien definida y sus capacidades avanzadas de procesamiento lo convierten en una herramienta clave en el ecosistema de procesamiento de streams.

2.2.3. Apache Flink

Apache Flink es una plataforma de procesamiento híbrida que ofrece soporte tanto para flujos de datos ("streams") como para procesamiento por lotes. Esta tecnología está desarrollada en Java y Scala y es ampliamente reconocida por su capacidad de manejar datos en tiempo real y de manera distribuida [1]. En su esencia, Flink se centra en un motor de flujo de datos distribuidos que procesa de manera uniforme trabajos tanto por lotes como de transmisión, los cuales están compuestos por tareas interconectadas con estado [42].

Características Clave de Flink

- Procesamiento híbrido: Flink soporta tanto flujos de datos (streams) como procesamiento por lotes. Su arquitectura está diseñada para ejecutar ambos tipos de cargas de trabajo de manera eficiente.
- APIs versátiles: Proporciona dos principales APIs:
  - La API *DataSet*, diseñada para el procesamiento de conjuntos de datos finitos (procesamiento por lotes).
  - La API *DataStream*, para flujos de datos potencialmente ilimitados (procesamiento de streams).
- Iteraciones nativas: Flink permite iteraciones dentro de sus flujos de datos, una funcionalidad útil para algoritmos de aprendizaje automático o procesamiento de grafos.
- Procesamiento con estado: Flink ofrece un manejo avanzado del estado de las aplicaciones, permitiendo almacenar y consultar datos intermedios durante el procesamiento de flujos.
- Entorno distribuido y local: Puede ejecutarse en clústeres distribuidos para escalabilidad o en un entorno local para facilitar el desarrollo y las pruebas.
- Tolerancia a fallos: Implementa un mecanismo de puntos de control ("checkpoints") que permite la recuperación rápida ante fallos, asegurando la consistencia del estado.
- Optimización automática: Incluye un optimizador que elige el mejor plan de ejecución según las características del entorno y los datos.
- Integración con ecosistemas existentes: Flink es compatible con gestores de clústeres como Apache Mesos, YARN, Kubernetes, y soluciones de almacenamiento como Apache Kafka, Apache Hadoop, HDFS, entre otros.
- Bibliotecas especializadas: Ofrece herramientas para tareas específicas como aprendizaje automático (FlinkML) y análisis de grafos (Gelly).

Arquitectura Flink

Apache Flink proporciona un entorno de ejecución unificado donde todos los programas se ejecutan. Estos programas se organizan como grafos dirigidos (JobGraphs) de operaciones paralelizadas, que también pueden incluir iteraciones [43]. Un JobGraph consiste en nodos y aristas; hay dos tipos de nodos: aquellos con estado (operadores) y aquellos lógicos (resultados intermedios o IRs). A continuación, se detallan los componentes de la arquitectura de Flink, como se muestra en la Figura 2.3:

- Librerias e Interfaces: En adición a la interfaz clásica de Flink, la librería **FlinkML** ofrece una serie de algoritmos y análisis de datos para machine learning. *Gelly* permite el análisis de grafos, mientras que *Table API* permite especificaciones declarativas de consultas similares a SQL [2].

- Stream Builder y Common API: Encargados de traducir entre el entorno de ejecución y las interfaces (API), mediante la transformación de grafos dirigidos de operaciones lógicas en programas genéricos de streams de datos que se ejecutan en su propio entorno de ejecución. La optimización automática de programas de flujo de datos esta incluida en este proceso. El optimizador integrado por ejemplo elige el mejor algoritmo de unión para cada caso de uso respectivo, con el usuario especificando únicamente una operación de unión abstracta [2].

- Gestor de Clúster y Almacenamiento (*Cluster Management and Storage*): Flink es compatible con un número de gestores de clúster y soluciones de almacenamiento, tales como Apache Tez [44], Apache Kafka [45], Apache HDFS [46] y Apache Hadoop YARN [40].

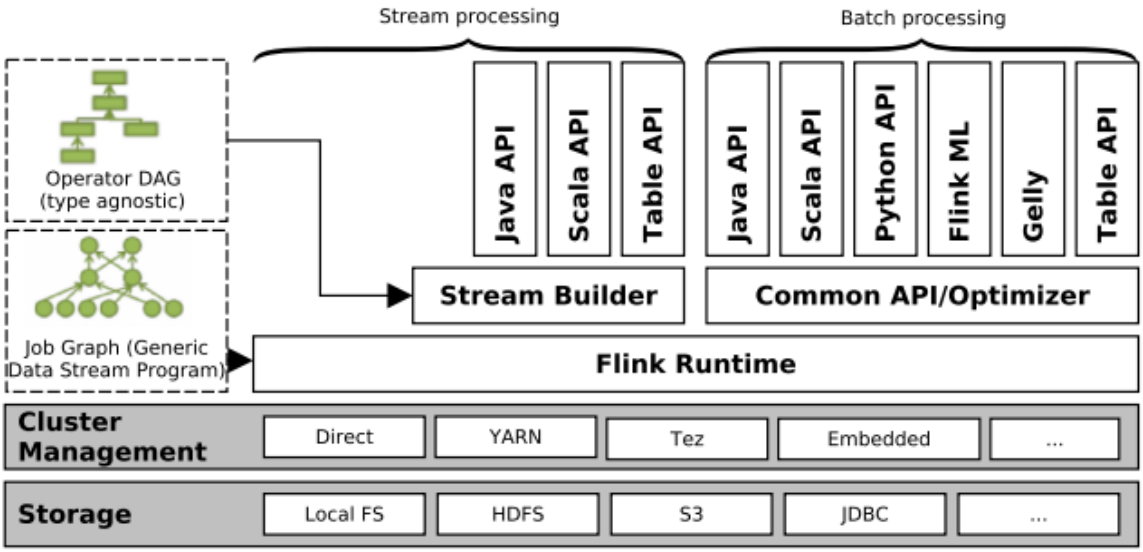

Figura 2.3: Arquitectura y componentes de la plataforma Apache Flink. Recuperado de [2]

Esta arquitectura propia de Flink posee similitudes a la propuesta por Storm, Flink utiliza un modelo maestro-trabajador. También se ha de diferenciar la arquitectura presentada con anterioridad a lo que es la arquitectura de un sistema Flink (ver Figura 2.4, esta última toma en cuenta los siguientes componentes:

- Gestor de Trabajos (*Job Manager*): Encargado de interactuar con las aplicaciones de los clientes y responsabilidades similares al nodo maestro de Storm. Recibe una serie de aplicaciones de clientes, organiza las tareas y las envía a los trabajadores. Además, se encarga de mantener el estado de todas las ejecuciones y los estados de cada trabajador.

- Gestor de Tareas (*Task Manager*): Ejecuta las tareas asignadas por el gestor de trabajos e intercambia información con otros trabajadores cuando es necesario. Cada gestor de tareas provee un número de ranuras de procesamiento a los clústers que son usados para ejecutar tareas en paralelo.

- Abstracción de Stream: Es llamada flujo de datos (DataStreams) y se define como secuencias de registros parcialmente ordenados. DataStreams pueden ser creados desde fuentes externas tales como colas de mensajes, socket streams, generadores personalizados o por la invocación de operaciones en otros DataStreams. DataStream posee soporte para una serie de operadores tales como *map, filtering, reduction* y pueden

ser paralelizados colocando instancias paralelas para ser ejecutadas en diferentes particiones de los respectivos streams, permitiendo la ejecución distribuida de transformaciones de streams.

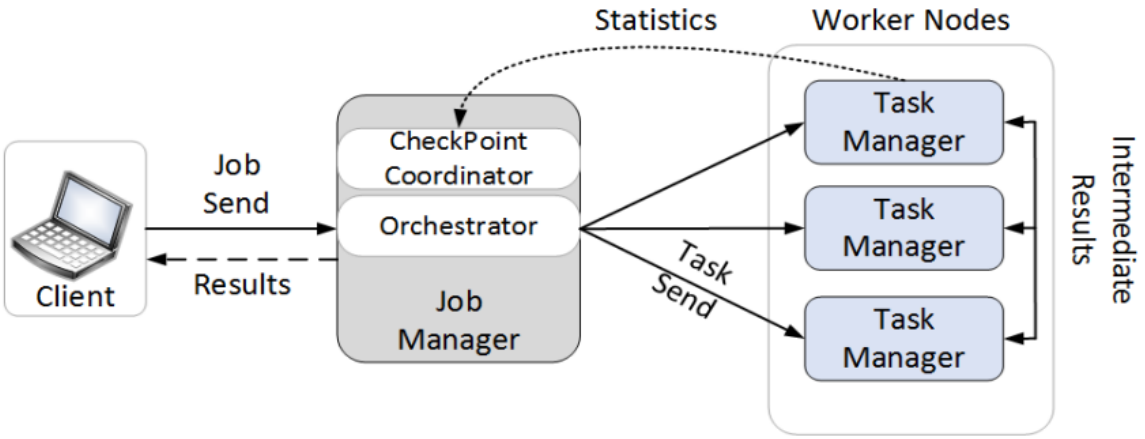

Figura 2.4: Arquitectura de un sistema Flink. Recuperado de [3]

Modelo de Ejecución El modelo de ejecución de Flink está basado en flujos de datos dirigidos (DataStreams), los cuales se definen como secuencias de registros que pueden ser procesados en paralelo. Sus principales características incluyen:

- Transformaciones de streams: Flink permite realizar operaciones como *map*, *filter*, *reduce*, *join*, entre otras, en flujos de datos.
- Paralelismo distribuido: Flink divide los flujos de datos en particiones, ejecutando instancias paralelas de operadores en diferentes nodos del clúster.
- Procesamiento con ventanas: Los flujos de datos pueden ser segmentados en "ventanas" para aplicar agregaciones y análisis en intervalos de tiempo específicos.
- Orquestación optimizada: El JobGraph generado por el usuario se traduce automáticamente en un plan de ejecución físico ("ExecutionGraph") que considera las capacidades del clúster y la distribución de los datos.

Fortalezas de Apache Flink

- Procesamiento híbrido: Flink permite combinar flujos y lotes en una misma aplicación, maximizando la flexibilidad.

- Manejo de estado: Su enfoque en el procesamiento con estado lo hace ideal para aplicaciones complejas que requieren consistencia.
- Compatibilidad: Se integra con una variedad de sistemas y entornos, incluyendo Kafka, HDFS, S3, Kubernetes y YARN.
- Tolerancia a fallos: Los puntos de control ("checkpoints") permiten recuperar el estado exacto de una aplicación en caso de fallo.
- Iteraciones eficientes: Es especialmente adecuado para algoritmos iterativos como los de aprendizaje automático.

Limitaciones

- Curva de aprendizaje: Su complejidad técnica puede ser desafiante para nuevos usuarios.
- Sobrecarga: En algunos casos, puede tener un rendimiento menor en comparación con soluciones especializadas en flujos o lotes.
- Configuración avanzada: Configurar Flink para clústeres distribuidos grandes puede ser complicado y requerir experiencia.

2.2.4. Apache Storm

Apache Storm es una plataforma de procesamiento distribuido para streams de datos en tiempo real. Desde sus inicios, Storm ha sido ampliamente reconocido y adoptado por algunas de las grandes industrias, tales como Twitter, Yahoo!, Alibaba, Groupon, entre otras [47]. Está diseñado para procesar y analizar grandes cantidades de streams de datos ilimitados, los cuales pueden provenir de diversas fuentes y publicar actualizaciones en tiempo real en la interfaz de usuario u otros lugares sin almacenar datos reales. Storm es altamente escalable y proporciona baja latencia con una interfaz fácil de usar a través de la cual los desarrolladores pueden programar virtualmente en cualquier lenguaje de programación [48]. Para lograr esta independencia de lenguaje, utiliza la definición de Thrift para definir e implementar topologías [1].

Características Clave de Apache Storm

- Storm es conocido por ser una plataforma flexible, escalable y de baja latencia para el procesamiento en tiempo real de grandes volúmenes de datos.
- Permite a los desarrolladores implementar soluciones para diversos casos de uso, como el procesamiento de streams de datos, sin necesidad de almacenamiento permanente de los mismos.

Arquitectura de Apache Storm

Apache Storm adopta una arquitectura maestro-esclavo, con un único nodo maestro y varios nodos esclavos. En la implementación física de Storm, se distinguen tres componentes principales (ver Figura 2.5):

- **Nimbus**: Nimbus es el demonio maestro que distribuye el trabajo entre todos los trabajadores disponibles. Sus principales responsabilidades incluyen el asignamiento de tareas a los nodos trabajadores, el seguimiento del progreso de las tareas, la reprogramación de las tareas para otros nodos trabajadores en caso de falla y la monitorización de las topologías que necesitan ser asignadas. Realiza la asignación entre esas topologías y los supervisores cuando es necesario [1].
- **Zookeeper**: Zookeeper actúa como coordinador entre Nimbus y los supervisores, ya que estos últimos no mantienen estado propio. Zookeeper es responsable de gestionar todo el estado necesario para el funcionamiento de la arquitectura [48].
- **Supervisor**: Los supervisores monitorean los procesos de cada topología e informan del estado a Nimbus utilizando el protocolo Heartbeat. Además, informan sobre la disponibilidad de ranuras (*slots*) disponibles que puedan asumir más trabajo.

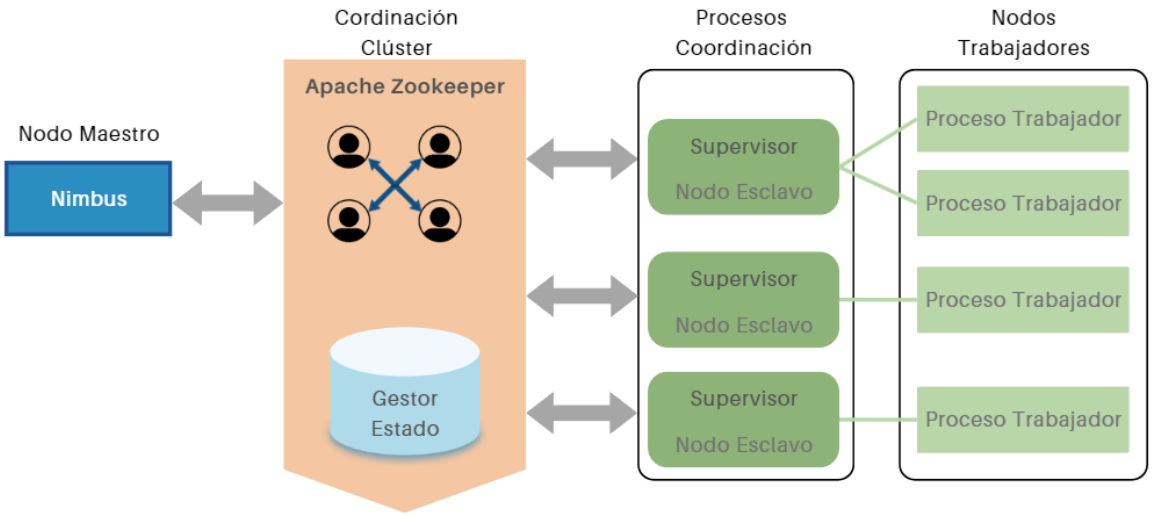

Figura 2.5: Arquitectura de Apache Storm. Recuperado de [1]

Modelo de Ejecución El modelo de procesamiento básico de datos de Storm consiste de 4 abstracciones: topología, spouts, bolts, stream. En Storm, un stream se define como una

secuencia de tuplas ilimitadas, donde las tuplas son listas nombradas de valores, que pueden ser de cualquier tipo, incluyendo cadenas de texto, enteros, números de punto flotante, etc. La lógica de cualquier aplicación Storm en tiempo real se presenta en forma de una topología, compuesta de una red de bolts y spouts (Ver Figura 2.6). Los Spouts son fuentes de stream que esencialmente se conectan con una fuente de datos como Kafka [45] o Kestrel. Estos reciben continuamente datos y los convierten en un flujo de tuplas para pasarlos a los bolts. Los bolts son las unidades de procesamiento de una aplicación de Storm que pueden realizar una variedad de tareas.

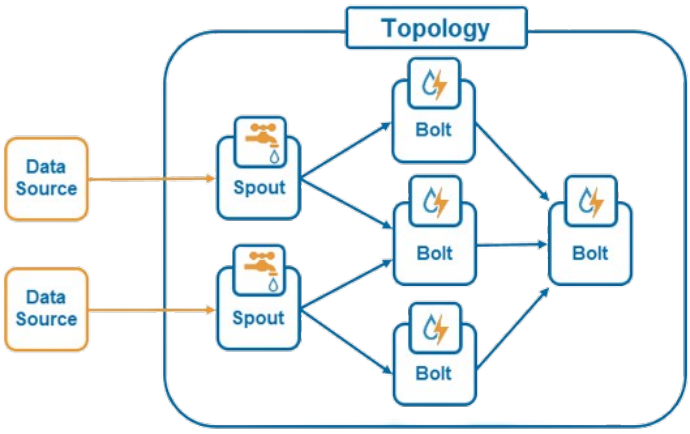

Figura 2.6: Topología base de una Aplicación Storm. Recuperado de [4]

.

Fortalezas de Apache Storm

- Altamente escalable, lo que le permite manejar grandes volúmenes de datos de manera eficiente.
- Capacidad de procesar datos en tiempo real con baja latencia.
- Flexibilidad para trabajar con diferentes lenguajes de programación lo hacen una herramienta poderosa para sistemas de análisis de datos en tiempo real.

Limitaciones

- Configuración y mantenimiento pueden resultar complejos en entornos de grandes dimensiones.
- El procesamiento de grandes volúmenes de datos requiere un monitoreo constante para asegurar que el sistema esté funcionando correctamente.

2.2.5. Apache Spark Streaming

Apache Spark es una plataforma de procesamiento de datos distribuidos, escrita en los lenguajes de programación Java y Scala. Incluye diversas bibliotecas de alto nivel que se ejecutan en el núcleo de Spark (*Spark Engine*), entre ellas Spark Streaming, que se utiliza para el procesamiento de streams de datos [7]. En Spark, la abstracción de streams se denomina *Stream Discreto* (*D-Stream*), definido como un conjunto de tareas cortas, sin estado y deterministas. El procesamiento de streaming en Spark se maneja como una serie de cómputos deterministas en lotes, procesando datos en pequeños intervalos de tiempo. Cuando un stream ingresa a Spark, este lo divide en micro-lotes, los cuales se tratan como datos de entrada en los conjuntos de datos resilientes (RDDs) [49]. Una vez procesados, estos micro-lotes se almacenan en memoria y se ejecutan trabajos para su procesamiento.

Arquitectura Spark

Apache Spark se compone de varios componentes esenciales y las bibliotecas de alto nivel. A continuación, se describe la arquitectura de Spark:

- **Librerías de alto nivel (*Libraries*)**: Se han construido varias bibliotecas sobre el núcleo de Spark para manejar diferentes cargas de trabajo, tales como: MLlib para machine learning [50], GraphX para procesamiento de gráficos [51, 52], Spark Streaming para análisis de streams [7], y Spark SQL para procesamiento de estructuras de datos [53].

- **Motor Spark (*Spark Core*)**: Spark Core proporciona una interfaz de programación simple para procesar grandes conjuntos de datos a gran escala. Este componente está implementado en Scala, pero cuenta con APIs en Java, Python y R. Además, ofrece funcionalidades clave para la computación de clúster en memoria, como la gestión de memoria, la planificación de trabajos, la redistribución de datos y la recuperación ante fallos [5].

- **Gestores de Clúster (*Cluster Management*)**: Un gestor de clúster administra los recursos del clúster para ejecutar tareas y manejar el intercambio de recursos entre aplicaciones Spark. Spark Core puede operar sobre varios gestores de clúster, como Hadoop YARN [54], Apache Mesos [41], Amazon EC2 y el gestor de clúster integrado de Spark.

- **Almacenamiento (*Storage*)**: Spark Core puede acceder a diversos sistemas de almacenamiento, como HDFS, Cassandra, HBase, Hive, Alluxio y algunas fuentes de datos de Hadoop [5].

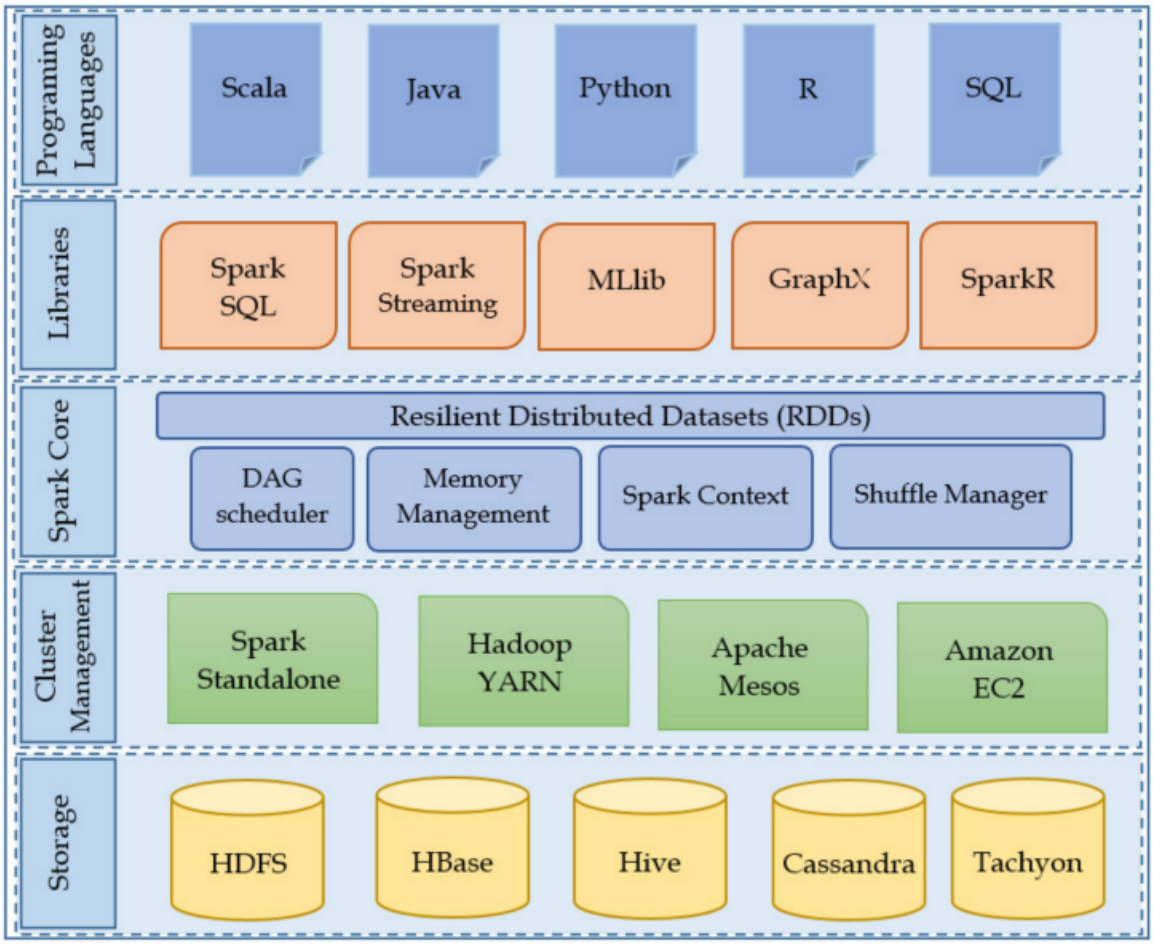

Figura 2.7: Arquitectura de alto nivel de Apache Spark. Recuperado de [5]

Aplicaciones en Spark Streaming

Para la ejecución de una aplicación de Spark se toman en cuenta los siguientes componentes clave (Ver Figura 2.8):

- Programa Controlador (*Driver Program*): Un programa controlador es una aplicación que utiliza Spark como una librería y contiene lógica de procesamiento de datos, este se encarga de crear el objeto *SparkContext* [5].
- Contexto de Spark (*SparkContext*): Cuando el SparkContext se ejecuta sobre un clúster, este tiene la capacidad de conectarse a varios tipos de gestores de clústers (ya sea el propio gestor de clúster independiente de Spark, Mesos, YARN o Kubernetes) y enviar tareas a los ejecutores [1].

- Ejecutores (*Executors*): Procesos JVM encargados de realizar cálculos y almacenar datos de la aplicación, cada aplicación obtiene sus propios procesos de ejecución que permanecen activos durante toda la duración de la aplicación y ejecutan tareas en múltiples hilos. Aislando las aplicaciones entre sí, tanto en el lado de la programación (cada controlador programa sus propias tareas) como en el lado del ejecutor (las tareas de diferentes aplicaciones se ejecutan en diferentes JVMs). Sin embargo, también significa que los datos no pueden compartirse entre diferentes aplicaciones de Spark (instancias de SparkContext) sin escribirlos en un sistema de almacenamiento externo [5].

- Nodo Trabajador (*Worker node*): Cualquier nodo que pueda ejecutar código de aplicación en el clúster. Permite alojar a los ejecutores y proveerlos de recursos de computación, tales como: CPU, memoria, y recursos de almacenamiento [1].

- Tarea (*Task*): Una tarea o task es la unidad de trabajo más pequeña que Spark envía a un ejecutor [5].

  La Figura 2.8 ilustra cómo se distribuyen estos componentes dentro de un clúster Spark, donde las aplicaciones se ejecutan como procesos independientes, sincronizados por el programa controlador, el contexto de Spark, los ejecutores, nodos trabajadores y tareas. A pesar de que Spark se basa en el intercambio de mensajes entre programas, a diferencia de Apache Storm, este enfoque introduce latencias adicionales, especialmente en aplicaciones con múltiples operaciones [3].

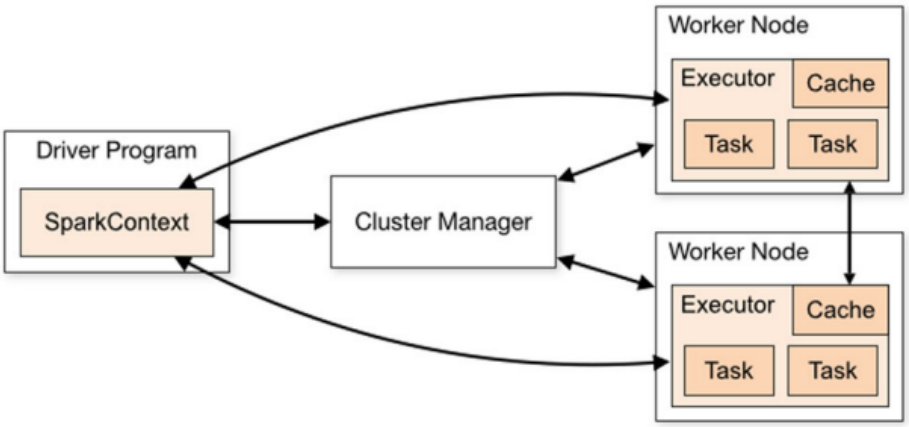

Figura 2.8: Entidades clave para ejecutar una aplicación Apache Spark. Recuperado de [6]

2.2.6. Trabajos Relacionados

A continuación, se presenta una recopilación de investigaciones previas relacionadas con la simulación de plataformas de procesamiento de streams:

Flow: Una plataforma de procesamiento de streams paralela y distribuida

La investigación realizada por **Park, S., et al.** nos presenta una plataforma paralela y distribuida (*Flow*) para simular aplicaciones de procesamiento de streams [55]. Este enfoque no incluye componentes de hardware o software del sistema de procesamiento de streams. En cambio, está diseñada de tal forma que es capaz de capturar el flujo de datos a nivel de aplicación.

Flow utiliza un enfoque de sincronización conservativa híbrida, destacándose por su capacidad para procesar eventos locales en orden de marca de tiempo inferior o LBTS (*Lower Bound Time Stamp*) y eventos que abarcan varios procesos a través de la sincronización conservativa. La plataforma aborda específicamente entornos de procesamiento de flujos en la nube, lo que la hace relevante para aplicaciones que manejan grandes volúmenes de datos en tiempo real.

RStorm: Desarrollo y prueba de algoritmos de streaming en R

Los autores **Muthukrishnan, P. y Fadnis** nos presentan una herramienta diseñada para el desarrollo y la prueba de algoritmos de procesamiento de datos en tiempo real utilizando el lenguaje de programación R, llamada RStorm [56]. RStorm es un paquete de simulación destinado a ayudar a los desarrolladores a analizar y evaluar sus algoritmos de streaming fácilmente y sin las dificultades de la implementación real en una plataforma de procesamiento de streams dada. RStorm recurre a la terminología y conceptos de Storm, proporcionando una representación gráfica de los algoritmos de streaming. Sin embargo, este paquete analiza el algoritmo y no incluye los costos de hardware asociados (como la comunicación, múltiples hilos por nodo, etc).

Simulación Paralela de Sistemas de Procesamiento de Streams Distribuidos a Gran Escala

En el artículo presentado por los autores **Fang, Z. y Yu, H.**, se aborda la problemática de simular sistemas de procesamiento de streams distribuidos a gran escala. Para abordar este desafío, los autores proponen un enfoque paralelo que aprovecha la computación distribuida para acelerar la simulación [57]. La arquitectura propuesta incluye componentes específicos para la generación de eventos, la sincronización y la comunicación entre nodos de simulación. Además de exponer técnicas para optimizar el rendimiento y la escalabilidad

de las simulaciones paralelas, como la partición del espacio de eventos y la minimización de la comunicación entre nodos.

Los autores **Ma, Y. y Rundensteiner, E. A.** abordan las mismas problemáticas presentes en la simulación de sistemas de procesamiento de streams distribuidos, pero desde una perspectiva centrada en la escalabilidad y eficiencia [58]. Este enfoque se basa en el uso de técnicas de muestreo adaptativo y la generación de muestras de eventos para la reducción de la carga computacional. Además, se presentan algoritmos para optimizar el rendimiento y la escalabilidad de las simulaciones distribuidas, como la adaptación dinámica de la tasa de muestreo y la paralelización de tareas.

Protocolo de simulación asíncrona para plataformas de procesamiento de streams

En el estudio llevado a cabo por **Gil-Costa, M., et al.**, se propone el uso de un protocolo de simulación asíncrona capaz de ejecutarse en la plataforma de procesamiento de streams S4 [59]. La simulación asíncrona es crucial para capturar con precisión el comportamiento de sistemas complejos que procesan flujos de datos en tiempo real.

Este simulador fue desarrollado para simular la ejecución de consultas de usuario en un motor de búsqueda web. El simulador propuesto controla el avance del tiempo virtual en cada Proceso Lógico o LP (*Logic Process*, por sus siglas en inglés), empleando dos barreras. La primera barrera consiste en una ventana de tiempo que permite el procesamiento de eventos con marcas de tiempo dentro de la ventana de tiempo. La segunda barrera se basa en una barrera de tiempo de oráculo utilizada para calcular de manera adaptativa el valor de la ventana de tiempo en toda la simulación. Los dos mecanismos basados en barreras reducen el número de eventos rezagados en una simulación asíncrona en todos los LP. Aunque el caso de estudio presente en esta investigación simula un motor de búsqueda web, los autores mencionan la posibilidad de modelar y simular una plataforma de procesamiento de streams utilizando este simulador.

Apache Heron: Baja Latencia en el Procesamiento de Streams

Los autores **Kamburugamuve, Supun, et. al.** proponen Apache Heron como un framework robusto y escalable de procesamiento de streamings distribuidos diseñado para manejar grandes volúmenes de datos en tiempo real [60]. Apache Heron propone una nueva arquitectura para el procesamiento de datos de streamings basada en un diseño híbrido con algunas partes críticas para el rendimiento escritas en C++ y otras escritas en Java. Esta arquitectura permite la integración de directa de mejoras de alto rendimiento en lugar de pasar por envoltorios (wrappers) nativos tales como la interfaz nativa de Java (JNI, por sus siglas en inglés). Los autores además abordan la necesidad del uso de una interfaz TCP para la interconexión de alto rendimiento, teniendo un desempeño bajo en comparación con una implementación nativa. Para resolver estos problemas de hardware Heron propone

la integración con Infiniband e Intel Omni-Path para interconectarlas con Apache Heron, con el fin de acelerar la comunicación.

Modelado y simulación de aplicaciones Spark y YARN utilizando lenguajes específicos de dominio

Los autores **Kross, D. y Krcmar, H.** abordan la necesidad de herramientas que permitan facilitar la representación y simulación de aplicaciones Spark y YARN en entornos de Big Data. En base a esta necesidad proponen un lenguaje específico de dominio o DSL(Domain-Specific Languages, por sus siglas en inglés) para modelar las características de las aplicaciones Spark y YARN. Al emplear DSL, se busca ofrecer una forma más intuitiva y eficiente de describir las características y el comportamiento de las aplicaciones, extrayendo automáticamente las especificaciones del modelo y transformándolas en una herramienta de evaluación. Mejorando significativamente el proceso de desarrollo y optimización de estas aplicaciones.

2.2.7. Comparativa de Plataformas de Procesamiento de Streams

En este análisis, se exploran varias plataformas líderes de procesamiento de streams y se evalúa su desempeño basado en una serie de métricas importantes a resaltar. A través de esta comparativa, se busca proporcionar una visión general completa de las características, ventajas y limitaciones de cada plataforma, lo que permitirá a los profesionales del área tomar decisiones informadas sobre cuál utilizar según sus necesidades específicas. El procesamiento de streams se ha convertido en una herramienta esencial en el mundo de la computación moderna, especialmente en aplicaciones que requieren el manejo de grandes volúmenes de datos en tiempo real, como las aplicaciones de análisis en vivo, sistemas de monitoreo y servicios de recomendación en plataformas de streaming.

El objetivo de este análisis es entender cómo las diferentes plataformas se posicionan frente a retos como la escalabilidad, latencia, facilidad de integración y robustez en entornos de producción. Además, se evalúan otras características clave como el soporte de múltiples lenguajes, la capacidad de manejar flujos de datos complejos y la eficiencia en el uso de recursos computacionales.

A continuación, se presenta una tabla resumen que abarca las plataformas más populares y utilizadas, propuestas por los autores **Khalid, Madiha y Yousaf, Muhammad Murtaza** [1] como parte de su investigación. En ella se comparan varias de las opciones más destacadas en el campo del procesamiento de streams, como Apache Kafka, Apache Flink, Apache Spark Streaming, y otras. La tabla no solo refleja el rendimiento de cada plataforma en términos de procesamiento de datos, sino que también ofrece una visión sobre la facilidad de uso, las opciones de integración con otras herramientas y servicios, y el nivel de soporte que ofrecen sus respectivas comunidades de usuarios. Con base en estos

criterios, se espera obtener una visión holística del panorama actual del procesamiento de streams.

Esta información será útil para las organizaciones que busquen implementar soluciones de procesamiento de streams eficientes, rentables y que puedan escalar según sus necesidades.

	Haddop	Spark	Storm	Samza	Flink
Principales Patrocinadores	Google Yahoo!	AMP Lab	Backtype Twitter	LinkedIn	dataArtisans
Lenguaje de Implementación	Java	scala	clojure	scala java	java scala
Soporte de Lenguaje de Programación	mayoría de lenguajes de alto nivel	java, scala python, R	cualquier lenguaje de programación	lenguajes JVM	java, scala python, R
Madurez	muy alta	alta	alta	media	baja

Tabla 2.1: Comparación de frameworks con respecto a sus características generales

	Haddop	Spark	Storm	Samza	Flink
Modelo Arquitectonico	maestro-esclavo	maestro-esclavo	maestro-esclavo	publicar-suscribir	maestro-esclavo
Gestor de Recursos	YARN	autónomo, YARN, Mesos	YARN, Mesos	autónomo, YARN, Mesos	autónomo, YARN, Mesos
Almacenamiento	HDFS	HDFS, HBase, Hive, Casandra	HDFS	HDFS	HDFS, streams database, todo en uno
Planificación	Justa, FIFO, Capacidad	Justa, FIFO	Defecto, aislamiento, multicliente, consciente de recursos	Planificador YARN	carga diferida, región de canalización
Seguridad	Protocolo de Autenticación Kerberos	Configuración basada en contraseña compartida, (ACLs)	Protocolo de Autenticación Kerberos	Sin seguridad integrada	Kerberos, autenticación TLS/SSL

Tabla 2.2: Comparación de frameworks con respecto a sus arquitecturas

	Hadoop	Spark	Storm	Samza	Flink
Formato de Ejecución	Solo batch	Batch y stream	Solo stream	Solo stream	Batch y stream
Modelo de Procesamiento de datos	MapReduce	DAG	Topología	Dag de operadores	Grafo de flujo de datos
Modo de Procesamiento	Procesamiento batch	Micro-lotes	Micro-lotes	Streaming nativo	Streaming nativo
Procesamiento en memoria	No	Sí	Sí	Sí	Sí

Tabla 2.3: Comparación de Frameworks con respecto al procesamiento de datos

	Spark	Storm	Samza	Flink
Garantías de Procesamiento	Exactamente una vez	Al menos una vez	Al menos una vez	Exactamente una vez
Fuente de datos	HDFS, DBMS, Kafka	Spout	Kafka	HDFS, DBMS, Kafka
Formato de Procesamiento	Micro-lotes	Micro-lotes	Flujo de streaming continuo	Flujo de streaming continuo, lotes, micro-lotes
Primitivas de Stream	Dstream	Tupla	Mensaje	Datastream
Gestión de Estado	Con estado	Sin estado	Operadores con estado	Operadores con estado

Tabla 2.4: Comparación de Frameworks con respecto al procesamiento de streams

Frameworks	Mejores casos de uso de aplicaciones
Hadoop	Aplicaciones que requieren el procesamiento por lotes de conjuntos de datos muy grandes donde el tiempo de ejecución no es una restricción estricta
Spark	Aplicaciones con cargas de trabajo por lotes o de streaming donde se prefiere un alto rendimiento más que una baja latencia
Storm	Aplicaciones de streaming donde se desea una latencia extremadamente baja con garantías de procesamiento al menos una vez
Samza	Aplicaciones de streaming que requieren que varios equipos accedan a los mismos flujos de datos en diferentes etapas de procesamiento
Flink	Aplicaciones con cargas de trabajo por lotes o de streaming donde se desea una latencia extremadamente baja con garantías de procesamiento exactamente una vez

Tabla 2.5: Casos de uso de aplicaciones donde cada uno de los frameworks se ajusta idealmente

En resumen, el análisis comparativo de las diversas plataformas de procesamiento de streams ha proporcionado una visión profunda de las fortalezas y debilidades de cada opción disponible en el mercado. A lo largo de esta evaluación, se ha podido observar que, aunque cada plataforma tiene sus características distintivas, la elección de la más adecuada depende en gran medida de las necesidades específicas de cada caso de uso. Factores como la latencia, la escalabilidad, el manejo de datos complejos y la facilidad de integración juegan un papel crucial en la toma de decisiones.

Es importante destacar que, si bien algunas plataformas, como Apache Kafka y Apache Flink, sobresalen en términos de rendimiento y capacidad de escalar en grandes volúmenes de datos en tiempo real, otras opciones como Apache Spark Streaming ofrecen una mayor flexibilidad en cuanto a los algoritmos de procesamiento y su integración con sistemas de análisis batch. Además, la elección de la plataforma debe tener en cuenta no solo el rendimiento técnico, sino también el ecosistema de soporte y las necesidades operativas, como la facilidad de mantenimiento, el soporte comunitario y la disponibilidad de herramientas complementarias.

En conclusión, la elección de la plataforma adecuada para el procesamiento de streams no es una decisión unívoca, sino que debe basarse en un análisis exhaustivo de los requerimientos del proyecto y las características específicas de cada plataforma. El panorama del procesamiento de streams continúa evolucionando, y con ello, las opciones disponibles seguirán mejorando en términos de capacidades, facilidad de uso y adaptabilidad a diferentes contextos.

Por lo tanto, se recomienda que las organizaciones realicen pruebas de concepto para validar cuál de estas plataformas se adapta mejor a sus necesidades operativas, asegurando así que la solución elegida no solo cumpla con los requerimientos técnicos, sino que también sea sostenible y escalable en el largo plazo.

Capítulo 3

Definición del Problema y Análisis de Requerimientos

El propósito de este capítulo es profundizar en el problema presentado en la sección anterior (véase la sección 1), proporcionando un nivel adicional de detalle. Posteriormente, se presenta la propuesta de solución y se analiza su relevancia en el ámbito científico y/o social.

3.1. Contexto del Problema

Las aplicaciones de simulación de plataformas de procesamiento de streams, como Apache Storm, desempeñan un papel crucial en la evaluación y optimización de sistemas de procesamiento de datos en tiempo real [61]. Estas herramientas permiten a los desarrolladores modelar y simular el comportamiento de diversas arquitecturas de procesamiento de streams en entornos controlados [62]. Al proporcionar una representación virtual de los componentes del sistema y sus interacciones, estas aplicaciones facilitan la identificación de cuellos de botella, la evaluación del rendimiento y la optimización del flujo de datos en tiempo real [63]. Además, su capacidad para manejar grandes volúmenes de datos y procesar eventos en tiempo real las convierte en herramientas indispensables para sectores como la analítica de datos, la IoT y la detección de anomalías [64]. En este contexto, exploraremos los desafíos y oportunidades asociados con el desarrollo, despliegue y escalado de aplicaciones tipo Apache Storm para la simulación de plataformas de procesamiento de streams.

Las aplicaciones de simulación de plataformas de procesamiento de streams se enfrentan a desafíos complejos en términos de escalabilidad y tolerancia a fallos. A medida que los sistemas de procesamiento de datos en tiempo real se vuelven más grandes y complejos, la capacidad de escalar horizontalmente y manejar flujos de datos de alta velocidad se convierte en una necesidad crítica. Además, la necesidad de garantizar la disponibilidad

continua y la integridad de los datos en entornos distribuidos plantea desafíos adicionales en términos de tolerancia a fallos y recuperación ante errores [65].

Uno de los principales desafíos en el desarrollo de aplicaciones Apache Storm es la complejidad asociada con el modelado y la implementación de topologías de procesamiento de streams eficientes. El diseño de topologías que puedan manejar flujos de datos de manera eficiente, distribuir la carga de trabajo de manera uniforme y adaptarse dinámicamente a cambios en los patrones de tráfico puede resultar complicado y requerir un profundo entendimiento de los principios de procesamiento de streams y las características del sistema subyacente [66].

Además de los desafíos técnicos, también existen consideraciones importantes relacionadas con la administración y el monitoreo de aplicaciones Apache Storm en producción. La capacidad de monitorear el rendimiento, la utilización de recursos y el estado de las topologías de procesamiento de streams en tiempo real es fundamental para garantizar la fiabilidad y el rendimiento óptimo del sistema. Por lo tanto, es crucial contar con herramientas y prácticas efectivas de monitoreo y gestión para identificar y mitigar problemas de manera proactiva, asegurando la disponibilidad continua de las aplicaciones críticas [67].

Estos desafíos plantean oportunidades significativas para la investigación y la innovación en el campo de las plataformas de procesamiento de streams. Al abordar los desafíos clave asociados con el desarrollo, despliegue y operación de aplicaciones Apache Storm, es posible mejorar la eficiencia, la escalabilidad y la confiabilidad de los sistemas de procesamiento de datos en tiempo real, lo que puede conducir a avances significativos en una amplia gama de aplicaciones y sectores industriales [68].

3.2. Definición del Problema

El presente trabajo se centra en abordar los desafíos identificados en la aplicación SimStream [69]. Esta aplicación, desarrollada en C++, se utiliza para la simulación y análisis de resultados a partir de una plataforma de procesamiento de streams en tiempo real, proporcionando una herramienta clave para modelar y estudiar sistemas de procesamiento de datos en entornos controlados. A pesar de su utilidad, durante la revisión literaria y el análisis de la aplicación existente, se identificaron una serie de áreas críticas que presentan limitaciones tecnológicas, las cuales pueden ser optimizadas para mejorar su desempeño y facilitar su integración en entornos de producción.

3.2.1. Complejidad de Configuración y Uso de Aplicación

La complejidad en la configuración y uso de una aplicación puede ser un obstáculo significativo para los usuarios, especialmente cuando la interfaz no es intuitiva o la documentación es insuficiente [70]. Cuando los usuarios enfrentan dificultades para configurar o

utilizar la aplicación de manera efectiva, esto puede limitar su utilidad práctica y afectar la adopción y satisfacción del usuario [71]. Por ejemplo, si un usuario encuentra dificultades para comprender la configuración de servicios o la interacción con la aplicación debido a una documentación poco clara, es probable que busque alternativas más sencillas y fáciles de usar, lo que puede reducir el número de usuarios activos de la herramienta.

3.2.2. Gestión eficiente del flujo de datos y comunicación entre componentes

La falta de un sistema gestor de colas de simulación puede ser un problema crítico en este contexto. Sistemas como RabbitMQ desempeñan un papel crucial en la gestión eficiente de los flujos de datos y la comunicación entre los distintos componentes de la aplicación [72]. Sin un sistema de este tipo, la aplicación puede enfrentarse a varios problemas relacionados con la sincronización de eventos, la gestión de la concurrencia y la escalabilidad. Además, la ausencia de un gestor de colas puede dificultar la implementación de patrones de diseño de software fundamentales, como el modelo productor-consumidor, lo que podría impactar negativamente en la eficiencia y fiabilidad del sistema en general [71]. En resumen, la carencia de un sistema gestor de colas de simulación limita las capacidades de procesamiento y escalabilidad de la aplicación, aumentando la complejidad del código y la dificultad en la gestión de los flujos de datos en tiempo real [70].

3.2.3. Portabilidad y Despliegue de Aplicación

El desarrollo de una aplicación de simulación de plataformas de procesamiento de streams en C++ conlleva desafíos considerables en cuanto a su portabilidad y despliegue, lo que puede dificultar la migración entre distintos sistemas operativos y plataformas de hardware [73]. La presencia de dependencias específicas del sistema y variaciones en las configuraciones de los entornos de desarrollo y producción agravan esta situación, limitando la flexibilidad y el alcance de la aplicación. La configuración manual de las dependencias y la gestión de versiones de software puede ser propensa a errores y consumir recursos, lo que impacta negativamente en la eficiencia del proceso de desarrollo y despliegue [74]. Estas dificultades, además, pueden obstaculizar la capacidad de la aplicación para adaptarse a las exigencias cambiantes de los usuarios y del mercado, limitando su escalabilidad y capacidad de respuesta ante nuevas demandas.

3.3. Solución Propuesta

Con el fin de abordar las problemáticas previamente mencionadas (ver secciones 3.2.1, 3.2.2, 3.2.3), se propone el desarrollo e implementación de un entorno web destinado a la simulación de plataformas de procesamiento de streams. El objetivo principal de este entorno es simplificar el proceso de creación y simulación de modelos de procesamiento de plataformas de streams, facilitando la interacción del usuario con el sistema. Este objetivo se logra a través de la abstracción y el encapsulamiento de las complejidades asociadas con la creación de modelos concurrentes o paralelos de plataformas de streams (ver Figura 3.1).

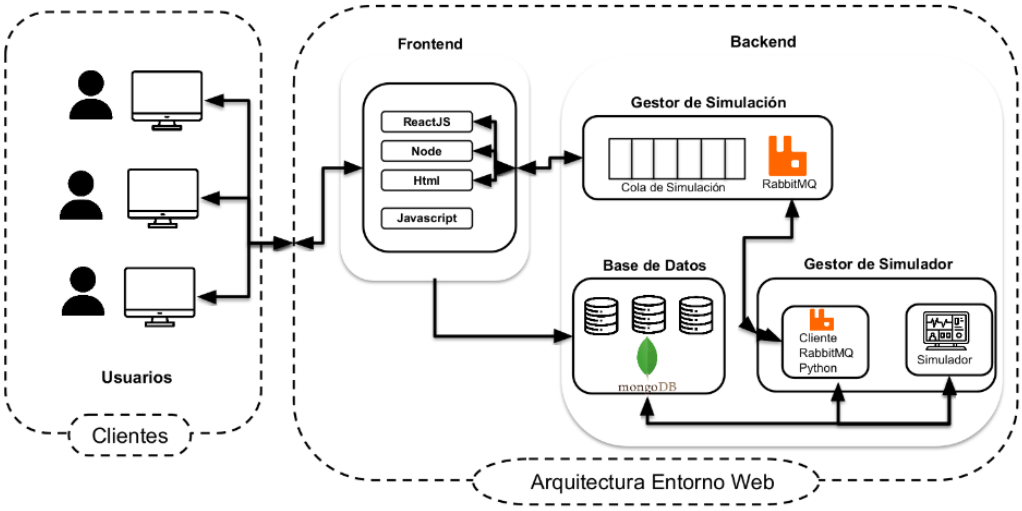

Figura 3.1: Arquitectura del entorno web para la simulación de plataformas de procesamiento de streams

La implementación de la solución propuesta es fundamental en la adopción de una arquitectura orientada a servicios (*SOA*, por sus siglas en inglés), en la que los componentes del sistema se estructuran como servicios independientes capaces de interactuar entre sí para cumplir tareas complejas. Este enfoque, ampliamente reconocido por su capacidad para facilitar la flexibilidad y escalabilidad del software, reduce las dependencias dentro del sistema al abstraer los detalles de implementación de cada servicio [75]. Además, permite que los servicios sean desarrollados en diversos lenguajes de programación y desplegados de forma modular, lo que facilita su mantenimiento y la integración de nuevas funcionalidades [76] (ver Figura 3.2).

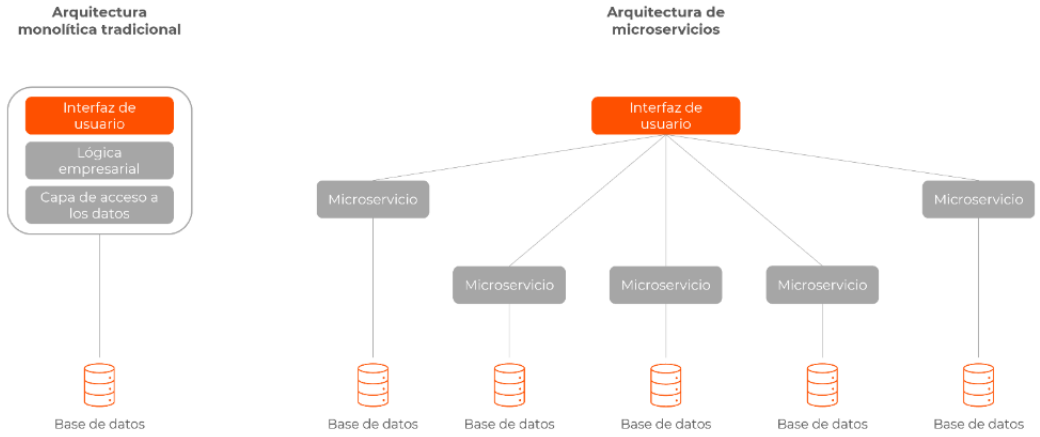

Figura 3.2: Arquitectura enfocada en microservicios

El enfoque adoptado en esta propuesta se centra en la creación de servicios que se ejecutan dentro de contenedores gestionados por la plataforma Docker [16]. Uno de los principales contenedores generados es el servicio de simulación de modelos de procesamiento de streams. Este servicio recibe como punto de entrada los modelos de simulación creados por los usuarios, ejecuta las simulaciones correspondientes y proporciona los resultados en un formato apto para análisis y validación.

Además, se implementará un servicio especializado para gestionar las colas de simulación creadas por los usuarios, así como para almacenar y recuperar los datos relacionados con los modelos de simulación y los usuarios del sistema.

De forma complementaria, se llevará a cabo un proceso de investigación y desarrollo para integrar una arquitectura orientada a eventos, utilizando el sistema de mensajería RabbitMQ [72]. Esta integración facilitará la comunicación asíncrona entre los diferentes componentes del sistema, permitiendo un manejo eficiente de los flujos de datos y tareas distribuidas.

Por otro lado, se utilizará **NestJS**, un marco robusto y escalable para la construcción de aplicaciones web y servicios API, que facilitará el desarrollo de la lógica del servidor y la gestión de las peticiones de los usuarios. NestJS permite la implementación de rutas parametrizadas para la gestión de operaciones CRUD (crear, leer, actualizar y eliminar), la validación de esquemas de solicitudes, así como el manejo de middleware para organizar la lógica de la aplicación de forma modular. Esto permitirá una gestión eficiente de las sesiones de usuario, así como la protección frente a posibles vulnerabilidades de seguridad en el sistema.

Adicionalmente, se implementarán pruebas exhaustivas de funcionalidad, rendimiento y usabilidad sobre el entorno web de simulación, en colaboración con modelos de simulación previamente validados [77]. Estas pruebas se centrarán en la obtención de métricas

comparativas en términos de eficiencia, rendimiento y uso de memoria, así como en la validación de la seguridad del sistema y su capacidad para manejar diferentes tipos de carga y escenarios de uso [77].

3.4. Objetivos

3.4.1. Objetivo General

- Diseñar, desarrollar e implementar un entorno web centrado en el uso de una arquitectura de servicios y el sistema de contenedores Docker como ejes principales, con el objetivo de resolver los problemas de complejidad presentes en la simulación de plataformas de procesamiento de streams, facilitando la creación, ejecución y análisis de modelos simulados de manera eficiente y escalable.

3.4.2. Objetivos Específicos

- Diseñar una arquitectura orientada a servicios adecuada para el entorno de simulación de plataformas de procesamiento de streams, con el objetivo de facilitar el desacoplamiento y la reutilización de componentes, asegurando flexibilidad y escalabilidad en el sistema.

- Asignar tareas específicas y bien delimitadas a cada servicio dentro del entorno de simulación, garantizando la independencia parcial de los servicios, así como su capacidad para comunicarse y orquestarse eficientemente en el sistema.

- Construir módulos que abarquen la interfaz de usuario, la gestión de simulaciones, la visualización de resultados y la integración con herramientas de terceros, asegurando una experiencia de usuario fluida y eficiente.

- Utilizar Docker como plataforma de contenedores para garantizar la portabilidad y consistencia del entorno de ejecución, asegurando que los entornos de desarrollo y producción sean fácilmente replicables y escalables.

- Crear un proceso de transformación que permita que el simulador SimStream opere como un servicio independiente, compatible con contenedores y escalable según las necesidades del sistema.

- Utilizar controladores y servicios para modularizar el código, diseñando un conjunto completo de rutas en NestJS que permita manejar operaciones específicas y garantice la seguridad mediante el uso de middleware y filtros de excepción.

- Gestionar operaciones CRUD a través de rutas parametrizadas en controladores NestJS, proporcionando una API que permita la manipulación de datos mediante HTTP, validando los datos con los pipes integrados de NestJS.
- Usar RabbitMQ como broker de mensajes para garantizar una distribución eficiente, confiable y asíncrona de mensajes entre los componentes del sistema, optimizando el flujo de comunicación y procesamiento de datos.
- Realizar pruebas exhaustivas de funcionalidad, rendimiento y usabilidad, utilizando casos de prueba representativos y métricas relevantes para medir la eficacia, confiabilidad y eficiencia del entorno en producción.
- Crear un informe técnico detallado que documente el proceso de diseño, desarrollo e implementación del entorno de simulación, incluyendo resultados obtenidos en funcionalidad, rendimiento y usabilidad, enriquecido con ejemplos prácticos, casos de uso y lecciones aprendidas.

3.5. Metodología

3.5.1. Metodología de Desarrollo

Para la ejecución de este trabajo de título, se adoptará la metodología de desarrollo ágil Scrum [17]. En Scrum, cada ciclo de desarrollo se organiza en sprints, que son períodos de tiempo fijos y cortos, generalmente de una a cuatro semanas de duración. Durante la planificación de cada sprint, el equipo selecciona una lista de elementos de trabajo del producto y se compromete a completarlos antes del final del sprint. Estos elementos de trabajo se desglosan en tareas más pequeñas y se estiman en función de la complejidad y el esfuerzo requerido.

Durante el sprint, el equipo se reúne diariamente en sesiones denominadas scrums diarios, donde cada miembro comparte su progreso, identifica posibles obstáculos y coordina las tareas restantes. Al finalizar el sprint, se realiza una retrospectiva, en la cual se revisa lo que se hizo bien, lo que se hizo tan bien y se identifican oportunidades de mejora para el siguiente sprint.

Este enfoque iterativo permite al equipo entregar un Producto Mínimo Viable (MVP) al término de cada sprint, proporcionando así un valor incremental y tangible al cliente [17] (ver Figura 3.3).

Figura 3.3: Metodología Desarrollo Scrum.

Después de varias iteraciones de sprints, se alcanza la fase de entrega, en la cual se presenta al cliente el producto funcional resultante de los sprints realizados [17]. Esta entrega incremental permite al cliente validar el producto en etapas tempranas y proporciona retroalimentación valiosa para orientar el desarrollo futuro.

Una vez entregado el producto, se inicia la fase de soporte y mantenimiento, durante la cual se brinda asistencia continua al cliente para garantizar el funcionamiento óptimo del producto en producción [78]. Esta fase incluye la corrección de errores, la aplicación de parches de seguridad y la implementación de mejoras y actualizaciones según sea necesario. Además, el equipo de desarrollo puede continuar trabajando en nuevos sprints con el objetivo de agregar funcionalidades adicionales o realizar mejoras en el producto existente.

A continuación, se presenta una figura que resume los conceptos discutidos anteriormente:

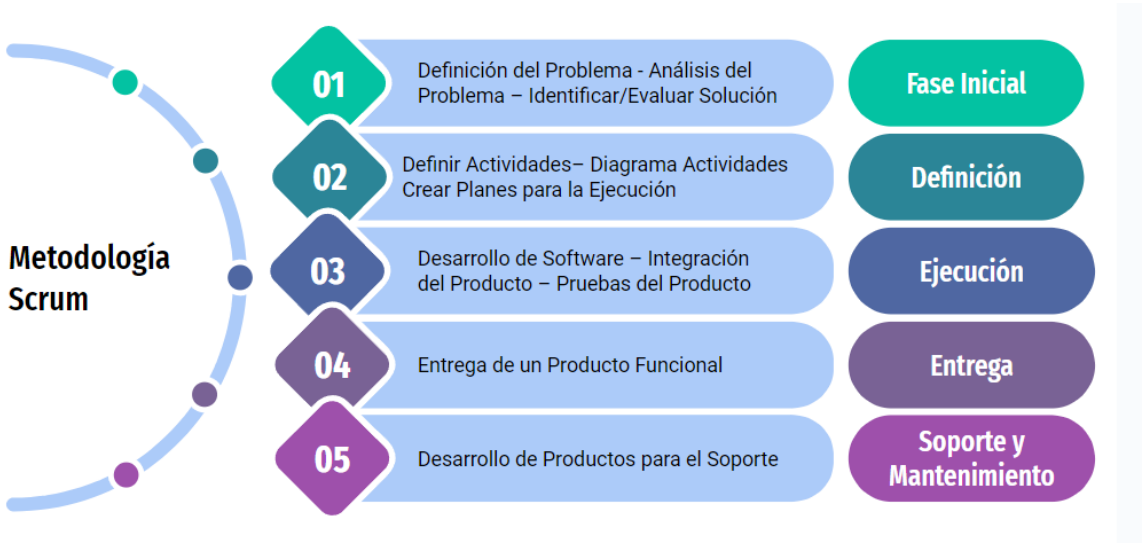

Figura 3.4: Fases de una Metodología Scrum.

3.5.2. Planificación de Desarrollo basada en Scrum

A continuación se presenta una planificación adaptada a la metodología de desarrollo Scrum. Esta planificación permite una entrega incremental de funcionalidades y mayor flexibilidad para adaptarse a los cambios en los requisitos del proyecto a lo largo del tiempo:

- Sprint 1: Definición del Alcance y Diseño Inicial
  - Definir los requisitos iniciales y el alcance del proyecto.
  - Realizar un análisis preliminar del dominio del problema y las necesidades de simulación.
  - Diseñar la arquitectura inicial del sistema, incluyendo la separación de responsabilidades entre servicios y los módulos principales.
  - Establecer las bases del repositorio y configurar el entorno de desarrollo, incluyendo Docker.
- Sprint 2: Implementación de la Arquitectura de Servicios
  - Implementar la arquitectura básica utilizando el marco de trabajo NestJS.
  - Configurar módulos iniciales, como los de RabbitMQ y la base de datos (MongoDB Atlas).

  - Crear servicios básicos y controladores que reflejen la estructura inicial del sistema.
  - Establecer integración inicial con Docker Compose para el despliegue de servicios.

- Sprint 3: Desarrollo de Componentes Clave
  - Desarrollar módulos específicos para la gestión de simulaciones y comunicación con RabbitMQ.
  - Implementar controladores y servicios relacionados con las rutas básicas de la API.
  - Validar la estructura de datos de simulación y garantizar su integridad durante el procesamiento.
  - Configurar la comunicación entre servicios independientes, como producer-service, consumer-service, simulation-service, bff, user-service.
- Sprint 4: Implementación del Entorno de Simulación
  - Diseñar e implementar procesos para consumir y procesar mensajes en el consumer-service.
  - Configurar Docker para el despliegue de simuladores en contenedores independientes.
  - Crear un flujo de trabajo funcional para simular el procesamiento de datos y enviar resultados a los servicios correspondientes.
  - Configurar colas de Dead Letter Queue (DLQ) para la gestión de errores.
- Sprint 5: Configuración y Gestión de Rutas
  - Desarrollar y configurar rutas parametrizadas en los controladores de NestJS.
  - Implementar la gestión de colas utilizando RabbitMQ para el intercambio de mensajes entre servicios.
  - Agregar validaciones básicas en el producer-service y la integración del manejo de errores en el consumer-service.
  - Asegurar que las rutas permitan operaciones CRUD en las simulaciones mediante controladores bien estructurados.
- Sprint 6: Pruebas y Validación
  - Realizar pruebas unitarias y de integración en los servicios desarrollados.

  - Evaluar la funcionalidad y rendimiento del entorno de simulación con casos de prueba representativos.
  - Validar la comunicación entre servicios y la consistencia de datos en la base de datos.
  - Garantizar la usabilidad y accesibilidad de la interfaz de usuario para el entorno de simulación.

- Sprint 7: Ajustes Finales y Documentación
  - Realizar ajustes finales en el sistema basados en los resultados de las pruebas.
  - Documentar la arquitectura, la implementación y los casos de uso del entorno de simulación.
  - Preparar el sistema para su entrega, incluyendo la revisión de la documentación técnica y la generación de archivos necesarios para los usuarios.

3.6. Especificación de Requerimientos

3.6.1. Requerimientos Funcionales

Los requerimientos funcionales son especificaciones detalladas que describen las funciones y comportamientos que un sistema debe ejecutar para cumplir con sus objetivos [79]. Estas especificaciones establecen no solo las acciones específicas que el sistema debe realizar en respuesta a las entradas proporcionadas por el usuario, sino también los resultados o salidas esperadas. En un contexto más amplio, los requerimientos funcionales pueden abarcar la descripción de procesos comerciales que el sistema debe automatizar o mejorar, así como casos de uso que modelan las interacciones típicas entre el usuario y el sistema [80]. Dichos requerimientos suelen representarse mediante historias de usuario, diagramas de flujo o casos de uso, sirviendo como base fundamental para el diseño, desarrollo e implementación del sistema.

Es fundamental señalar que los requerimientos funcionales deben ser claros, precisos y verificables [79]. Estos deben ser comprensibles tanto para los desarrolladores responsables de la implementación del sistema como para los clientes o usuarios finales que interactuarán con él. Asimismo, los requerimientos funcionales deben mantener un grado de flexibilidad, permitiendo adaptaciones y modificaciones a medida que evolucionan las necesidades del negocio o se identifican nuevos requisitos durante el proceso de desarrollo. En este contexto, a continuación se presentan algunos de los requerimientos funcionales establecidos para este trabajo de título:

- Inicio de Sesión:

  - El sistema permitirá a los usuarios iniciar sesión utilizando un nombre de usuario y una contraseña válidos.
  - Se verificará la autenticidad de las credenciales proporcionadas antes de otorgar acceso al sistema.

- Registro de Usuarios:
  - Los usuarios podrán registrarse proporcionando información básica, como nombre, dirección de correo electrónico y contraseña.
  - El sistema validará que la dirección de correo electrónico proporcionada sea única dentro de la base de datos.

- Dashboard de Usuario:
  - Después de iniciar sesión, los usuarios serán redirigidos a un tablero (dashboard) que les permitirá realizar diversas operaciones.
  - Este dashboard ofrecerá la funcionalidad de realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) sobre los modelos de simulación de plataformas de streaming.

- Operaciones CRUD sobre Modelos de Simulación:
  - Los usuarios podrán crear nuevos modelos de simulación proporcionando la información requerida.
  - Será posible visualizar una lista de todos los modelos de simulación existentes, así como editarlos si es necesario.
  - Los usuarios tendrán la opción de eliminar modelos de simulación que ya no sean requeridos.

- Almacenamiento en MongoDB:
  - Todos los datos de usuario, incluidos los detalles de registro y la información de los modelos de simulación, se almacenarán en bases de datos MongoDB.
  - Esto incluirá los resultados de las simulaciones realizadas, los cuales estarán disponibles para su posterior recuperación, procesamiento y análisis.

- Servicio de Gestión de Colas de Simulación:
  - El sistema interactuará con un servicio de gestión de colas de simulación implementado mediante RabbitMQ.
  - Los modelos de simulación creados por los usuarios serán enviados a la cola de simulación para su procesamiento.

- Servicio de Simulación Desarrollado en C++
    - El sistema incluirá un servicio de simulación desarrollado en C++, modificado para ser ejecutado como un contenedor de Docker.
    - Este servicio será responsable de procesar los modelos de simulación provenientes de la cola de simulación y de generar las simulaciones y resultados correspondientes.

3.6.2. Requerimientos No Funcionales

Los requerimientos no funcionales, también denominados atributos o características de calidad, especifican las cualidades globales que un sistema debe poseer más allá de sus funcionalidades específicas. Estos requerimientos son esenciales para garantizar que el sistema no solo cumpla con sus objetivos funcionales, sino que también sea eficiente, seguro y adaptable a su entorno operativo [81]. Entre los aspectos más relevantes que abordan los requerimientos no funcionales se incluyen:

Estos requerimientos no funcionales son tan importantes como los funcionales, ya que influyen directamente en la experiencia del usuario, la eficiencia del sistema y su capacidad para satisfacer los objetivos del negocio. Aunque suelen ser más abstractos y difíciles de medir, son esenciales para garantizar que el proyecto de software sea exitoso y sostenible a largo plazo [79]. A continuación, se presentan los requerimientos no funcionales identificados para este trabajo de título:

- Seguridad:
    - Los datos de los usuarios y los modelos de simulación deben almacenarse de forma segura y protegidos contra accesos no autorizados.
    - El sistema debe implementar medidas de seguridad, como cifrado de contraseñas y protección contra ataques de inyección de código.
- Rendimiento:
    - El sistema debe ser capaz de manejar una carga de trabajo significativa sin sufrir degradación en el rendimiento.
    - Las operaciones CRUD y la simulación de modelos de stream deben ejecutarse de manera eficiente, asegurando tiempos de respuesta aceptables.
- Escalabilidad:
    - El sistema debe ser escalable horizontalmente para gestionar el aumento en la cantidad de usuarios y modelos de simulación.

    - Debe permitir la fácil adición de nuevos recursos, como instancias o servicios en contenedores Docker, para incrementar su capacidad operativa.

- Disponibilidad:
    - El sistema debe estar disponible las 24 horas del día, los 7 días de la semana, con un tiempo mínimo de inactividad programado para tareas de mantenimiento.
    - Se deben implementar medidas de redundancia y respaldo para garantizar la continuidad de los datos y servicios críticos.
- Interfaz de Usuario:
    - La interfaz de usuario, desarrollada en React JS, debe ser intuitiva y fácil de usar, con un diseño atractivo y adaptable (responsive) a diferentes dispositivos y tamaños de pantalla.
    - Debe proporcionar una experiencia fluida para enviar los modelos de simulación creados por los usuarios a la cola de simulación para su procesamiento.
- Compatibilidad:
    - El sistema debe ser compatible con una amplia gama de navegadores web modernos, como Google Chrome, Mozilla Firefox, Safari y Microsoft Edge.
    - La aplicación debe ser funcional en distintos sistemas operativos, como Windows, macOS y Linux.
- Mantenibilidad:
    - El código del sistema debe seguir buenas prácticas de desarrollo, ser legible y modular, y estar adecuadamente documentado para facilitar su mantenimiento y futuras actualizaciones.

3.7. Funcionalidades del Sistema

La sección de funcionalidades del sistema proporciona una visión detallada de las capacidades y características ofrecidas por la arquitectura del entorno web para la simulación de plataformas de procesamiento de streams. Desde la gestión de usuarios hasta la generación de informes, este sistema ha sido diseñado para satisfacer una amplia variedad de necesidades. Entre las principales funcionalidades se incluyen la creación y edición de perfiles de usuario [82], la administración de contenido [83], la programación de eventos [84], la generación de informes y el análisis de datos [85], y la integración con sistemas externos [86]. Cada funcionalidad ha sido desarrollada con el objetivo de facilitar el uso del

sistema y adaptarse a las necesidades específicas de los usuarios. A continuación, se detallan las funcionalidades clave que convierten a este sistema en una herramienta poderosa y versátil para la gestión y análisis de datos.

3.7.1. Diagramas de Casos de Uso

Actores

- **Usuario**: El actor ”Usuario” representa a cualquier persona que acceda al sistema como cliente. Este usuario tiene la capacidad de iniciar sesión en el sistema para acceder a sus funcionalidades, registrarse como nuevo usuario si no tiene una cuenta, y gestionar los modelos de simulación de plataformas de stream. Esto incluye crear, guardar, eliminar y visualizar detalles de los modelos existentes. Además, puede generar informes detallados sobre los resultados de las simulaciones realizadas.

- **Administrador**: El actor ”Administrador” es responsable de administrar y supervisar el funcionamiento del sistema. Tiene las mismas capacidades que un usuario normal, como iniciar sesión y gestionar modelos de simulación, pero además cuenta con funcionalidades adicionales. El administrador puede simular modelos de manera directa, gestionar usuarios (ver lista de usuarios, editar información de usuarios existentes, desactivar o eliminar cuentas de usuarios si es necesario), y realizar cualquier acción necesaria para garantizar el correcto funcionamiento del sistema y la seguridad de los datos.

3.7.2. Casos de Uso

A continuación, se presenta el diagrama de caso de uso de la solución propuesta (ver Figura 3.5). Este diagrama proporciona una visión general de las funcionalidades clave que ofrece el sistema y cómo interactúan los actores con el mismo para lograr sus objetivos. El diagrama está estructurado en dos partes principales: ”Gestión de Modelos de Simulación” y ”Administración del Sistema”.

En el caso de uso ”Gestión de Modelos de Simulación”, tanto los usuarios normales como los administradores tienen la capacidad de realizar diversas acciones, tales como iniciar sesión, registrarse, gestionar modelos de simulación y generar informes basados en los resultados de las simulaciones realizadas. Es importante señalar que el caso de uso denominado ”Simular Modelo” está directamente relacionado con la creación de modelos de simulación y la carga de estos para su posterior simulación.

Por otro lado, el caso de uso ”Administración del Sistema” está reservado exclusivamente para los usuarios administradores. Estos tienen la responsabilidad de gestionar a los usuarios y administrar el sistema en su conjunto. Esto incluye acciones como la gestión de la información de los usuarios, así como la realización de tareas administrativas

relacionadas con la configuración y el mantenimiento del sistema.

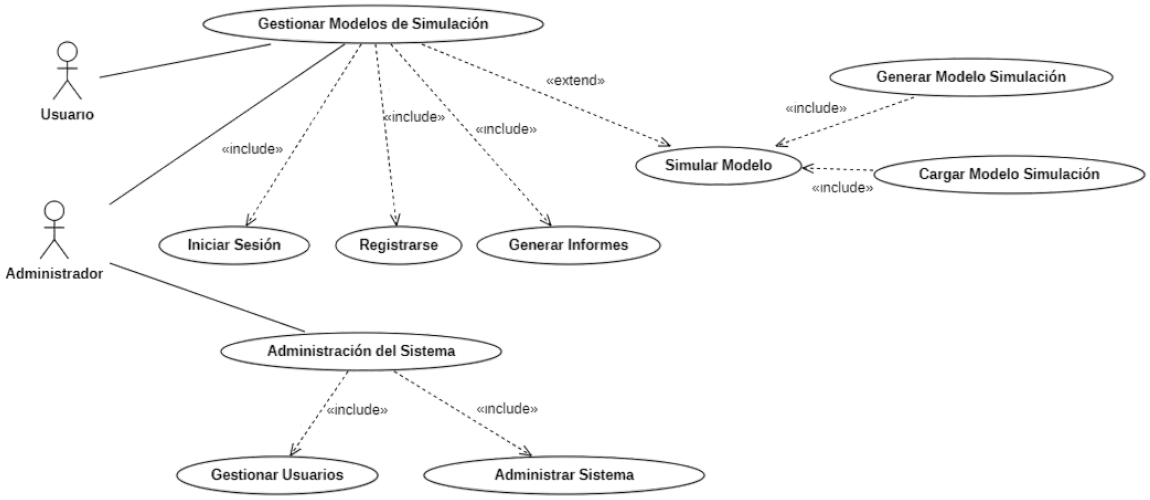

Figura 3.5: Diagrama General de Caso de Uso.

3.7.3. Diagramas de Estado

A continuación se presenta el diagrama de estado para usuarios (Ver Figura 3.6), en donde se muestran los estados y transiciones posibles del sistema, basado en el caso de uso anteriormente proporcionado (3.7.1). Los estados incluyen lo siguientes elementos:

- Usuario No Autenticado
- Usuario Autenticado
- Gestionar Modelos
- Simulando Modelo
- Finalizado

Las transiciones entre estados están etiquetadas en conjunto con las acciones que desencadenan un cambio de estado, como iniciar sesión, cerrar sesión, simular modelos, generar informes, y otras acciones relacionadas con la gestión de modelos y simulaciones pertinentes.

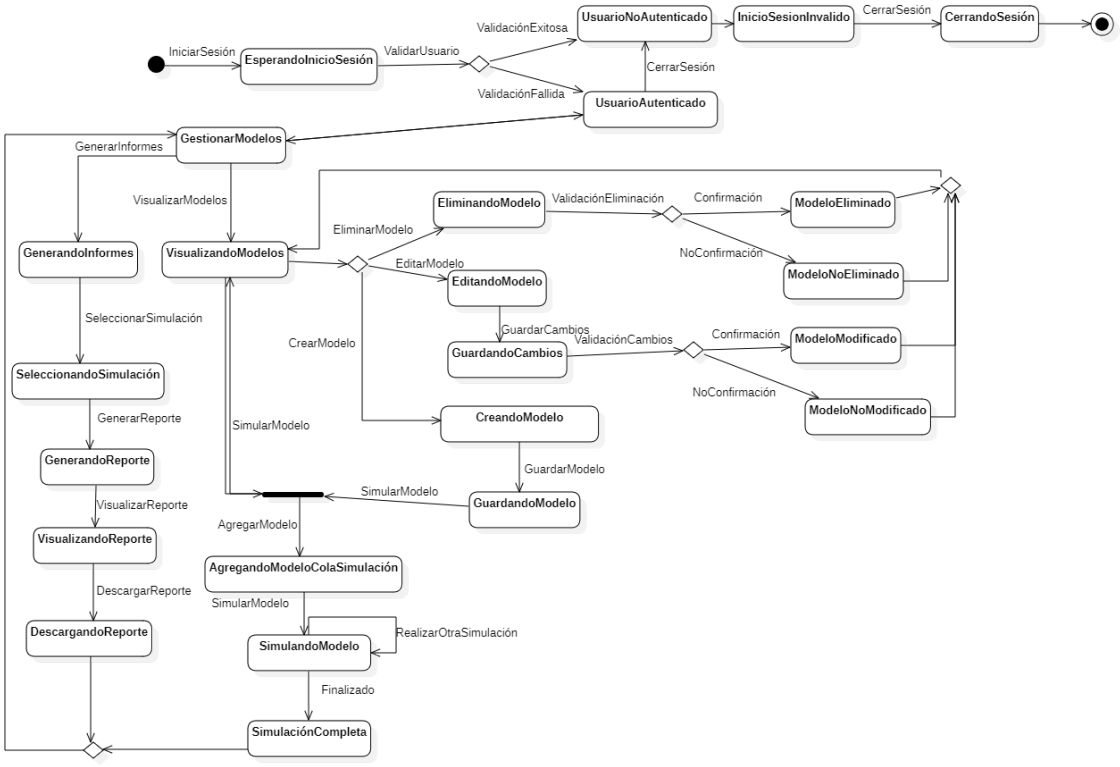

Figura 3.6: Diagrama de estado para usuarios.

3.7.4. Modelo Conceptual

El modelo conceptual del proyecto refleja la estructura general del sistema, así como las entidades clave y sus interacciones para gestionar las simulaciones, los resultados, y los estados de las simulaciones. Este modelo es fundamental para entender cómo los diferentes componentes del sistema interactúan entre sí, asegurando que los flujos de datos y la comunicación entre servicios sean claros y efectivos.

En el núcleo del sistema, tenemos varias entidades esenciales que gestionan las simulaciones y su ciclo de vida. Estas entidades incluyen:

1. Usuario: Representa a los usuarios del sistema que crean y gestionan las simulaciones. Los usuarios pueden tener múltiples simulaciones asociadas a su cuenta, y cada simulación está vinculada a un conjunto de nodos y resultados.

2. Simulación: Es la entidad central del sistema, que representa la ejecución de una simulación que genera resultados. Cada simulación puede tener varios nodos asociados

y un estado que puede cambiar a lo largo de su ciclo de vida, desde su creación hasta su finalización. La simulación puede pasar por diversos estados como 'pendiente', 'en ejecución', 'completada', o 'fallida'.

3. Nodo: Un nodo es un componente dentro de la simulación que realiza una tarea específica. Los nodos pueden ser de diferentes tipos, como nodos de tipo S (simulación) o de tipo B (nodo final). Los nodos pueden generar datos como parte de su proceso, que se utiliza para calcular los resultados de la simulación.

4. Resultado: Los resultados contienen la salida generada por la simulación, que puede incluir métricas o cualquier otro dato relevante. Los resultados están asociados con una simulación específica y pueden ser actualizados a medida que la simulación avanza.

5. Estado de Simulación: Cada simulación tiene un historial de estados que refleja su progreso y la fase en la que se encuentra. Estos estados se registran para mantener un seguimiento detallado de las simulaciones.

Este modelo conceptual también define las relaciones entre las entidades de la siguiente forma:

- Un usuario puede crear muchas simulaciones.
- Una simulación puede tener múltiples nodos y está asociada con un solo resultado generado por el simulador SimStream.
- El estado de simulación cambia con el tiempo y se registra para cada simulación.

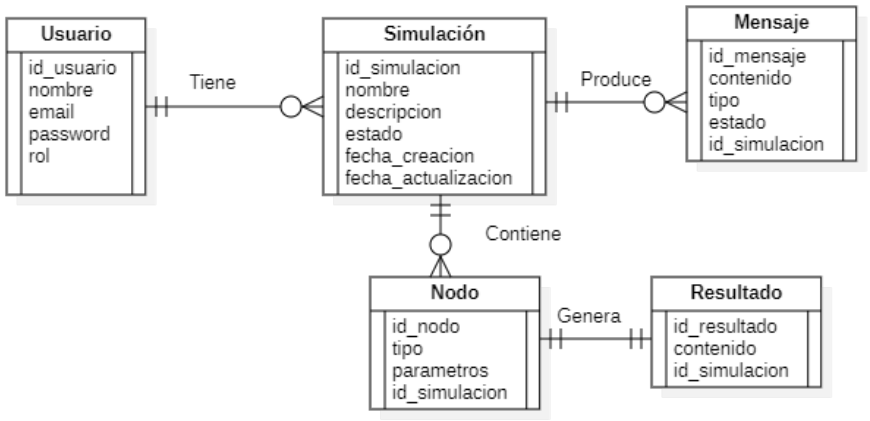

Figura 3.7: Modelo Conceptual del entorno web

Capítulo 4

Diseño de la solución

Este capítulo aborda en detalle la solución propuesta para el problema planteado, incluyendo los objetivos generales y específicos, la propuesta de solución, las formulaciones utilizadas con sus respectivas explicaciones, los algoritmos diseñados y la metodología de investigación aplicada.

4.1. Diseño Arquitectónico

4.1.1. Tecnologías propuestas

A continuación, se presenta la asociación de cada uno de los requerimientos no funcionales (ver sección 3.6.2) con las tecnologías correspondientes utilizadas para el desarrollo del trabajo de título. Estas tecnologías han sido seleccionadas cuidadosamente con el fin de abordar cada uno de los aspectos clave identificados en los requerimientos, garantizando así un sistema seguro, eficiente, escalable, disponible, fácil de usar, compatible y fácil de mantener en el tiempo. Cada tecnología desempeña un papel crucial en la creación de una solución integral que cumpla con las expectativas de los usuarios y los requerimientos de negocio.

Seguridad

La seguridad es uno de los pilares fundamentales del sistema, tanto en la protección de los datos como en la protección de los usuarios y servicios. Para asegurar un sistema robusto y seguro, se utilizaron las siguiente tecnologías:

- Docker: Docker proporciona un entorno seguro y aislado para ejecutar las aplicaciones. Los contenedores son independientes y garantizan que las dependencias y

configuraciones del sistema no afecten a otro servicios. Además, Docker permite implementar políticas de seguridad de red, como la restricción de acceso entre contenedores y la habilitación de autenticación basada en certificados para la comunicación entre servicios [87]. También facilita el uso de herramientas como Docker Content Trust para garantizar la integridad de las imágenes desplegadas.

- RabbitMQ: RabbitMQ es una solución de mensajería robusta que también incluye políticas de seguridad integradas. Se puede configurar para que solo los usuarios autorizados tengan acceso a las colas de mensajes mediante la autenticación basada en contraseñas, certificados o tokens [88]. Además, permite la implementación de políticas de encriptación TLS para asegurar que los mensajes se transmitan de forma segura entre los servicios.
- NestJS: NestJS es un framework de Node.js que facilita la creación de aplicaciones escalables y seguras. Implementa herramientas y estrategias para proteger las aplicaciones, como la validación de entradas, la protección contra ataques CSRF, y el uso de JWT (JSON Web Tokens) para la autenticación [89]. Esto se complementa con módulos de seguridad integrados que permiten proteger las rutas y manejar las credenciales de forma segura.
- MongoDB: MongoDB implementa autenticación y autorización basada en roles, lo que garantiza que solo los usuarios con permisos adecuados puedan acceder o modificar los datos. MongoDB Atlas, la versión administrada en la nube, ofrece medidas avanzadas de seguridad, como la encriptación de datos en reposo y en tránsito, y configuraciones de acceso seguro mediante IP y autenticación de múltiples factores [90].

Rendimiento

El rendimiento es esencial para que el sistema pueda manejar grandes cantidades de usuarios y operaciones simultáneas sin perder eficiencia. Las tecnologías implementadas permiten optimizar cada parte del sistema:

- Docker: Docker permite ejecutar aplicaciones en contenedores ligeros, lo que reduce el overhead asociado con la virtualización tradicional. Esto resulta en tiempos de inicio rápidos y una gestión eficiente de los recursos [91]. Docker también facilita el escalado dinámico, permitiendo añadir instancias de contenedores cuando la carga lo requiere, manteniendo el rendimiento del sistema.
- RabbitMQ: RabbitMQ es altamente eficiente en la distribución de mensajes y permite el balanceo de carga entre múltiples instancias de consumidores, lo que optimiza el rendimiento bajo altas cargas. La implementación de ”exchanges” y colas duraderas asegura la alta disponibilidad y el procesamiento eficiente de los mensajes, incluso en escenarios de alta concurrencia [92].

- NestJS: NestJS se basa en Node.js, lo que le permite manejar operaciones asíncronas y no bloqueantes. Esto es clave para el rendimiento, ya que permite al sistema procesar múltiples solicitudes simultáneamente sin que se interrumpa el flujo de trabajo. Las aplicaciones construidas con Node.js son especialmente eficientes en la gestión de conexiones concurrentes gracias a su modelo de Event Loop [93].

Escalabilidad

La escalabilidad es crucial para permitir que el sistema crezca en función de la demanda. Las tecnologías seleccionadas permiten escalar tanto vertical como horizontalmente, según las necesidades del sistema:

- Docker: Docker facilita la escalabilidad horizontal, ya que permite crear y gestionar contenedores de manera eficiente. Esto permite la implementación de nuevas instancias del sistema para distribuir la carga de trabajo, lo que mejora la capacidad de respuesta y el manejo de más usuarios sin problemas de rendimiento [94].
- NestJS: Node.js es un entorno altamente escalable, ya que su modelo de E/S no bloqueante permite gestionar miles de conexiones simultáneas sin afectar el rendimiento del sistema. NestJS, como framework sobre Node.js, hereda estas propiedades, proporcionando un marco ideal para construir aplicaciones escalables con un enfoque modular que permite añadir nuevas funcionalidades sin afectar la performance [95].

Disponibilidad

La disponibilidad del sistema es clave para garantizar que el servicio esté siempre accesible. Las tecnologías implementadas aseguran que el sistema sea resiliente y que pueda recuperarse rápidamente de fallos:

- Docker: Docker facilita la implementación de prácticas de alta disponibilidad al permitir la replicación de contenedores y la distribución de servicios entre diferentes nodos. Esto garantiza que si un contenedor falla, otro pueda asumir su función sin interrupciones significativas [96].
- RabbitMQ: RabbitMQ garantiza la alta disponibilidad mediante la replicación de colas entre nodos. La configuración de clústeres de RabbitMQ permite que los mensajes sean distribuidos y almacenados en múltiples nodos, lo que reduce el riesgo de pérdida de datos y mejora la resiliencia ante fallos [97].

Compatibilidad

La compatibilidad con diferentes plataformas y tecnologías es esencial para asegurar que el sistema pueda interactuar con otros servicios y clientes:

- Docker: Docker mejora la compatibilidad al proporcionar un entorno homogéneo para las aplicaciones, independientemente del sistema operativo en el que se ejecuten. Esto asegura que las aplicaciones funcionen de la misma manera en diferentes entornos (desarrollo, pruebas, producción) y en diferentes infraestructuras [98].
- RabbitMQ: Al ser un sistema de mensajería basado en estándares abiertos, RabbitMQ permite la comunicación eficiente entre componentes del sistema, incluso si están construidos con diferentes tecnologías. Esto garantiza que el sistema pueda integrarse fácilmente con otros servicios, independientemente de su implementación [99].
- Express JS: Express JS permite la creación de APIs RESTful que siguen los estándares web abiertos, lo que facilita la interoperabilidad con una amplia variedad de clientes y otros servicios [100].

Mantenimiento

El mantenimiento a largo plazo es crucial para garantizar que el sistema pueda evolucionar con facilidad sin generar problemas técnicos. Las tecnologías seleccionadas permiten gestionar y mantener el sistema de forma eficiente:

- Docker: Docker facilita la gestión de versiones y dependencias de forma centralizada. Al ser un entorno reproducible, facilita las actualizaciones y el despliegue de nuevas versiones sin afectar a la infraestructura existente [101].
- MongoDB: MongoDB es una base de datos flexible que permite realizar cambios en el esquema de datos de forma rápida y sencilla, lo que facilita la adaptación a nuevos requisitos del negocio sin complicaciones [102].
- Express JS: La arquitectura modular de Express JS permite la reutilización de código, lo que facilita la implementación de nuevas funcionalidades y la gestión del sistema a medida que evoluciona.

4.2. Diseño Lógico

4.2.1. Diagrama de despliegue

En esta sección se presenta el Diagrama de Despliegue del proyecto de título (ver Figura 4.1), que muestra la distribución física de los componentes y servicios del sistema,

así como las interacciones entre ellos. El diagrama proporciona una vista general de cómo se despliega la aplicación, las conexiones entre los servicios y los mecanismos utilizados para garantizar la comunicación eficiente y segura entre ellos.

El sistema sigue una arquitectura basada en microservicios desarrollados con NestJS, donde cada servicio está estructurado en módulos y servicios específicos que encapsulan tareas particulares y facilitan la separación de responsabilidades. Todos los componentes están desplegados en contenedores Docker, interconectados a través de una red interna definida en Docker Compose. A continuación, se describen los principales componentes y sus interacciones:

- Frontend (React.js / Next.js): Representa la interfaz de usuario, permitiendo a los usuarios interactuar con el sistema. Este contenedor se comunica exclusivamente con el BFF a través de una network interna de Docker, asegurando que el acceso esté restringido y solo permitido entre servicios autorizados.

- Backend For Frontend (BFF): Es el intermediario entre el frontend y los demás microservicios. Se encarga de recibir las solicitudes del usuario, validarlas y redirigirlas a los servicios correspondientes, como el user-service, producer-service o simulation-service.

- User Service: Servicio NestJS responsable de la gestión de usuarios, autenticación y autorización. Está estructurado en módulos que gestionan validaciones, lógica de negocio y consultas a la base de datos MongoDB Atlas.

- Producer Service: Servicio NestJS que recibe solicitudes relacionadas con la creación de simulaciones. Este servicio está dividido en módulos específicos para procesamiento de datos, validación, creación y envío de mensajes a la cola principal de RabbitMQ mediante el módulo de mensajería.En caso de fallos, los mensajes son redirigidos a la Dead Letter Queue (DLQ) para ser consumidos y actualizados los estados de los modelos de simulación a un estado 'failed'.

- Consumer Service: Servicio NestJS encargado de consumir mensajes de la cola principal de RabbitMQ. Está organizado en módulos que procesan los mensajes, validan los datos enviados, ejecutan simulaciones y actualizan el estado en la base de datos MongoDB Atlas a través de la comunicación con el Simulation Service.

- Simulation Service: Servicio NestJS que gestiona toda la lógica relacionada con las simulaciones, almacenamiento de modelos de simulación, resultados y actualizaciones en MongoDB Atlas. Está compuesto por módulos de controladores HTTP, lógica de negocio y acceso a la base de datos.

- RabbitMQ: Sistema de mensajería que facilita la comunicación asíncrona entre los servicios NestJS. RabbitMQ incluye:

    - Cola Principal: Almacena los mensajes enviados por el Producer Service y los entrega al Consumer Service.
    - Dead Letter Queue (DLQ): Almacena mensajes que no pudieron ser procesados exitosamente, permitiendo su análisis, recuperación y posterior consumo para actualizar los estados de los modelos de simulación fallidos.

- MongoDB Atlas: Base de datos en la nube utilizada para almacenar información de usuarios, simulaciones y resultados. Los servicios NestJS de User Service y Simulation Service interactúan con la base de datos a través de módulos específicos de acceso y validación.

- Red Interna de Docker: Todos los servicios utilizan una red interna de Docker para garantizar una comunicación segura y controlada. Esta red permite que:

    - El FrontEnd solo pueda comunicarse con el BFF.
    - El BFF se comunique exclusivamente con los demás servicios mediante solicitudes HTTP.
    - Los microservicios se comuniquen entre sí o con RabbitMQ a través de rutas internas.

Flujo de Comunicación entre Componentes

La comunicación entre los componentes se realiza mediante los siguientes mecanismos:

- El Frontend envía solicitudes HTTP únicamente al BFF, que actúa como punto de entrada principal al sistema.

- El BFF redirige las solicitudes HTTP a los microservicios correspondientes: User Service, Producer Service o Simulation Service.

- El Producer Service envía mensajes a la cola principal de RabbitMQ, utilizando el módulo de mensajería de NestJS.

- El Consumer Service consume los mensajes de la cola, procesa los datos a través de sus módulos internos y actualiza el estado de las simulaciones en MongoDB Atlas.

- En caso de errores, los mensajes se redirigen automáticamente a la DLQ para ser analizados posteriormente.

- Todos los servicios NestJS utilizan módulos específicos para interactuar con MongoDB Atlas y RabbitMQ, siguiendo una arquitectura modular y desacoplada.

El Diagrama de Despliegue refleja una arquitectura distribuida y segura, donde los servicios desarrollados en NestJS están organizados en módulos especializados que garantizan la separación de responsabilidades. La comunicación controlada mediante la red interna de Docker y el uso de RabbitMQ como sistema de mensajería aseguran una interacción eficiente y resiliente entre los componentes del sistema.

En la Figura 4.1, se presenta un ejemplo de un diagrama de despliegue.

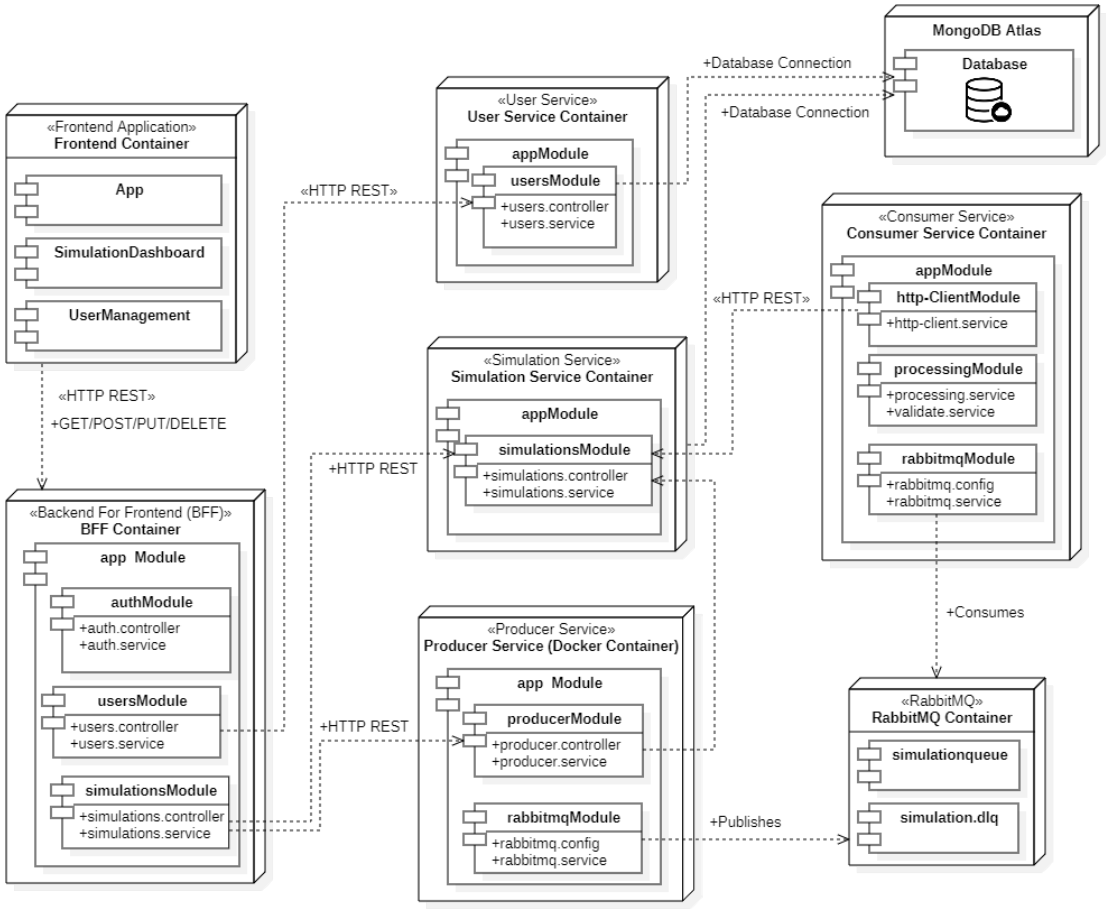

Figura 4.1: Diagrama de despliegue del entorno web

4.2.2. Diagrama de componentes

El diagrama de componentes que se presenta a continuación (ver Figura 4.2) proporciona una visión detallada de los servicios y sus interacciones dentro de la arquitectura de microservicios de este proyecto. En este diagrama, se ilustran los componentes principales

que forman parte del sistema, incluyendo el Frontend, el BFF (Backend For Frontend), los servicios de User Service, Simulation Service, Producer Service, Consumer Service, así como la base de datos MongoDB Atlas y el Broker de Mensajería RabbitMQ:

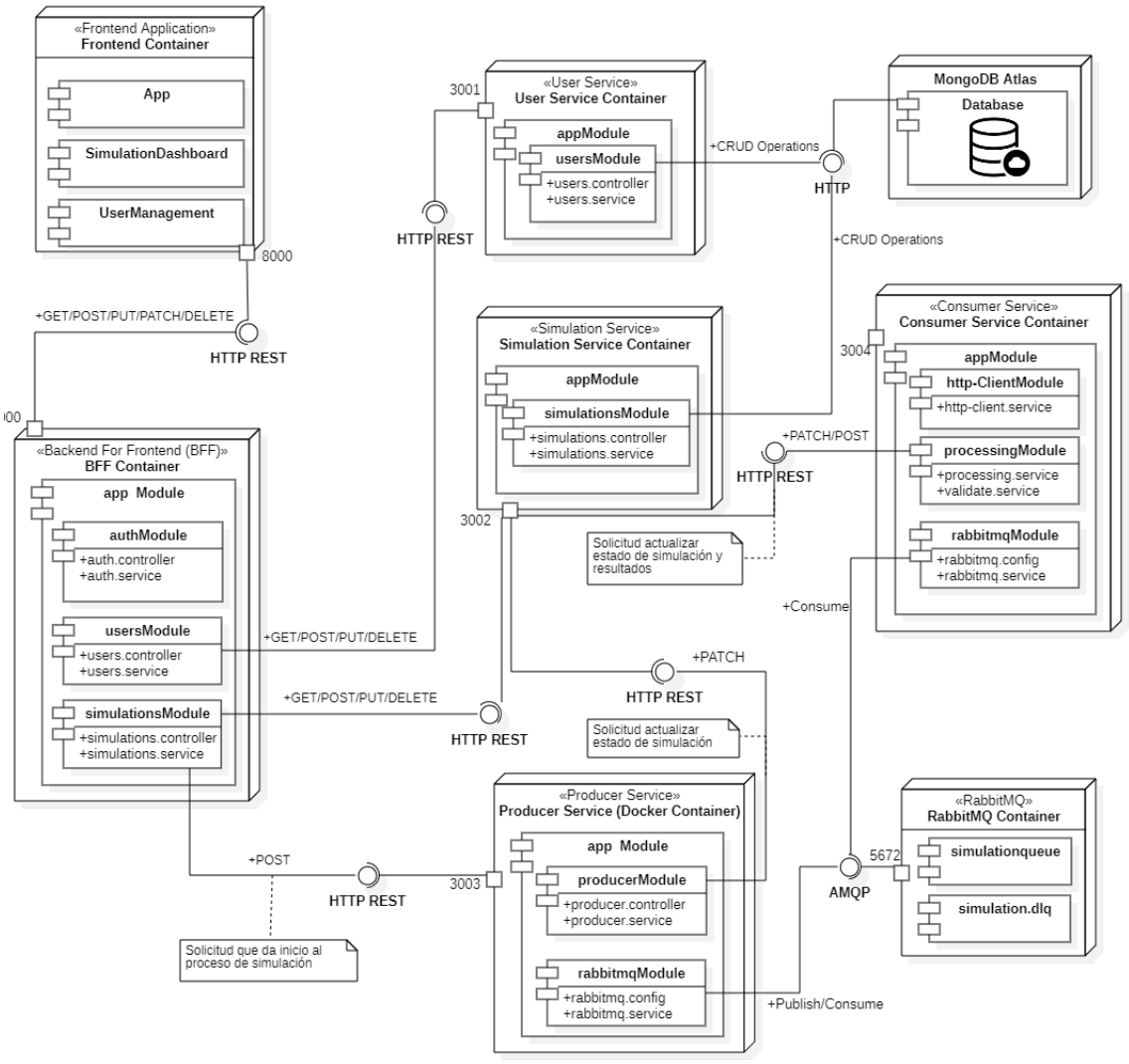

Figura 4.2: Diagrama de Componentes

Cada servicio está representado como un componente que interactúa con otros a través de diversas relaciones y protocolos, tales como HTTP REST, AMQP (para RabbitMQ) y conexiones de base de datos para realizar operaciones CRUD. A continuación, se explica cada uno de los componentes y sus relaciones:

- Frontend: Representa la interfaz de usuario que interactúa con el sistema a través del BFF. El usuario puede acceder a diversas funcionalidades del sistema, como la visualización de simulaciones, el inicio de nuevas simulaciones, y la visualización de resultados. La comunicación entre el Frontend y el BFF se realiza a través de solicitudes HTTP REST utilizando la red interna de Docker definida en el docker-compose.

- BFF (Backend For Frontend): Responsable de recibir las solicitudes del Frontend y dirigirlas a los servicios correspondientes, como User Service, Simulation Service, Producer Service y Consumer Service. Utiliza HTTP REST para comunicarse con estos servicios y, a su vez, interactúa con ellos para proporcionar las respuestas necesarias al Frontend. La comunicación entre estos componentes también está protegida por la red interna del contenedor de Docker.

- User Service y Simulation Service: El User Service y el Simulation Service interactúan con la base de datos MongoDB Atlas, donde almacenan y gestionan los datos correspondientes. Cada uno de estos servicios tiene su propia colección en MongoDB y realiza operaciones CRUD para la gestión de usuarios y simulaciones, respectivamente. La comunicación con la base de datos se realiza a través de un repositorio que se conecta a MongoDB Atlas.

- User Service y Simulation Service: Servicios que interactúan con la base de datos MongoDB Atlas, donde almacenan y gestionan los datos correspondientes. Cada uno de estos servicios tiene su propia colección en MongoDB y realiza operaciones CRUD para la gestión de usuarios y simulaciones, respectivamente. La comunicación con la base de datos se realiza a través de un repositorio que se conecta a MongoDB Atlas.

    - User Service maneja la creación, actualización y eliminación de usuarios, así como la recuperación de sus datos.
    - Simulation Service realiza las mismas operaciones CRUD pero sobre las simulaciones, gestionando el ciclo de vida de cada simulación.

- Producer Service: Responsable de generar mensajes que se publican en el Broker de Mensajería RabbitMQ. Este servicio establece una conexión a RabbitMQ utilizando el protocolo AMQP y crea las colas necesarias, como la cola principal y la cola de mensajes muertos (DLQ). El Producer Service también es el encargado de encolar los mensajes en la cola principal para que sean procesados por el Consumer Service.

- Consumer Service: Responsable de consumir los mensajes que llegan a través de RabbitMQ desde el Producer Service. Una vez que el mensaje es procesado, el servicio puede realizar las acciones necesarias, como la ejecución de simulaciones o el

procesamiento de los resultados, y luego actualizar el estado de la simulación a través del Simulation Service utilizando HTTP REST.

- RabbitMQ: Actúa como el Broker de Mensajería en este sistema. Gestiona la comunicación asíncrona entre el Producer Service y el Consumer Service, permitiendo que los mensajes se publiquen en la cola principal o en la DLQ (Dead Letter Queue) para su posterior procesamiento. El Producer Service es el encargado de crear las colas y gestionar la encolación de los mensajes. El Consumer Service, por su parte, se suscribe a estas colas para consumir los mensajes y procesarlos de acuerdo con la lógica de negocio.
- MongoDB Atlas: Base de datos en la que se almacenan los datos del sistema, y está dividida en dos colecciones principales: simulations y users. El User Service interactúa con la colección users para gestionar los datos de los usuarios, mientras que el Simulation Service maneja la colección simulations para almacenar y recuperar las simulaciones.

Este diagrama de componentes proporciona una visión clara de cómo los distintos servicios dentro del sistema interactúan entre sí y con las infraestructuras externas, como la base de datos y RabbitMQ. Cada servicio tiene responsabilidades claras y está desacoplado de los demás, lo que permite una mayor flexibilidad y escalabilidad dentro del sistema. La comunicación entre los componentes se realiza a través de interfaces bien definidas, utilizando tecnologías como HTTP REST, AMQP y operaciones CRUD en la base de datos MongoDB Atlas.

4.2.3. Diagrama de paquetes

El diagrama de paquetes es una representación visual que muestra cómo los diferentes módulos y componentes del sistema están organizados y cómo interactúan entre sí dentro de la arquitectura de la aplicación. En el contexto de nuestro proyecto, donde se implementan diversos microservicios utilizando NestJS y contenedores Docker, el diagrama de paquetes ofrece una visión estructurada de cómo se agrupan los servicios y sus dependencias internas.

Este diagrama es fundamental para entender la organización de los distintos servicios dentro del sistema, como el BFF (Backend For Frontend), el User Service, el Simulation Service, el Producer Service, el Consumer Service, y las interacciones con componentes externos como RabbitMQ y MongoDB Atlas. Cada uno de estos servicios tiene un conjunto de responsabilidades claramente definidas, y la representación en paquetes permite visualizar cómo estos interactúan entre sí de manera modular, ayudando a mejorar la comprensión de la arquitectura general del sistema.

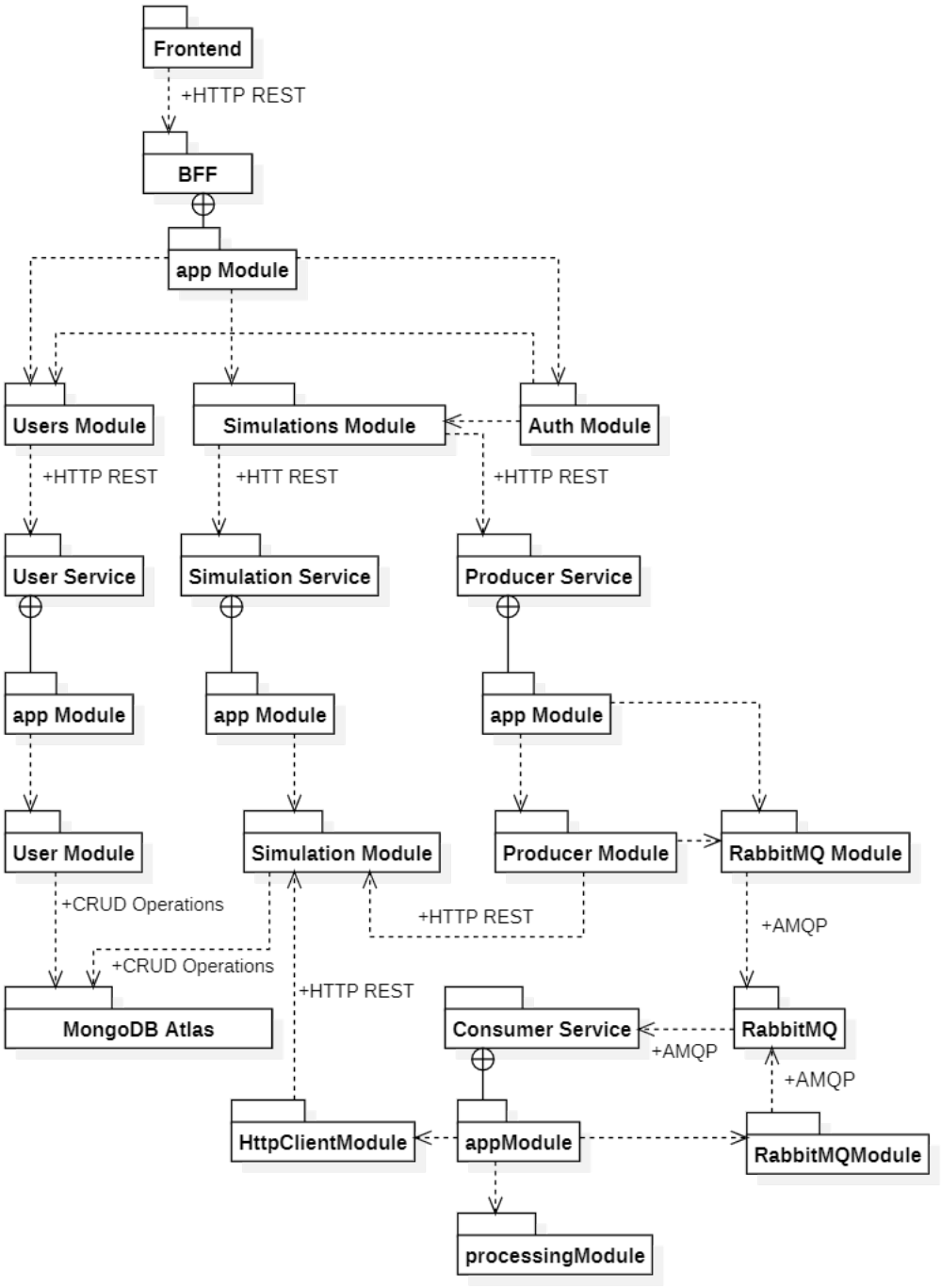

Figura 4.3: Diagrama de Paquetes del entorno web

4.2.4. Diagrama de clases

BFF

El BFF (Backend for Frontend) es un componente fundamental en la arquitectura de la aplicación que se encarga de recibir las solicitudes del frontend y redirigirlas a los servicios adecuados para su procesamiento. Su responsabilidad principal es actuar como una capa intermedia que adapta las peticiones del cliente (generalmente un frontend) a los servicios y APIs del backend, gestionando la interacción entre los diversos microservicios y proporcionando datos estructurados de forma coherente para el frontend.

- Bootstrap: Hace referencia al proceso en el que se inicializa el servicio y se configuran los módulos necesarios para su funcionamiento. Se invoca al método NestFactory.create() para crear la instancia de la aplicación y ejecutar el método app.listen() para iniciar el servidor.

- appModule: Módulo raíz responsable de encapsular toda la configuración del servicio, incluyendo otro módulos. Aquí se importan los módulos necesarios para la funcionalidad de la aplicación, como el SimulationsModule, ConfigModule para cargar las variables de entorno, el AuthModule y el UserModule.

- SimulationsModule: Módulo encargado de agrupar todos los componentes necesarios para gestionar los modelos de simulación. Este módulo define la estructura y organización de los servicios, controladores y entidades relacionados con las simulaciones. Facilitando la inyección de dependencias, configuración de rutas y servicios necesarios para crear, actualizar y consultar modelos de simulación en el sistema. Este modulo también se encarga de realizar la validación de roles, los tokens de los usuarios que intenten comunicarse con los demás servicios del backend y en especial acá se redirecciona y valida todas las peticiones provenientes del frontend.

- SimulationsService: : Clase responsable de gestionar la lógica de negocio relacionada con las simulaciones. Este servicio maneja las operaciones de creación, actualización y consulta de simulaciones, así como las excepciones relacionadas con ellas, tales como errores de validación y problemas internos del servidor. Su principal responsabilidad es validar las solicitudes y redirigirlas al Simulation Service y Producer Service. Además, gestiona la comunicación entre el frontend y los demás servicios, asegurando que las solicitudes se manejen de manera eficiente y adecuada.

- UsersService: Clase responsable de gestionar la lógica de negocio relacionada con los usuarios. Este servicio maneja las operaciones de creación, actualización, consulta y eliminación de usuarios, así como las excepciones asociadas, como errores de validación y problemas internos del servidor. Su principal responsabilidad es validar

las solicitudes y redirigirlas a los servicios correspondientes, como User Service y Authentication Service. Además, gestiona la comunicación entre el frontend y los demás servicios, asegurando que las solicitudes de los usuarios sean manejadas de manera eficiente y segura.

Nos centraremos principalmente en los componentes y modulos responsables de todo el proceso de autenticación y autorización que se implementó en el BFF, n la arquitectura del backend, el BFF (Backend for Frontend) actúa como intermediario entre el frontend y los servicios backend, gestionando la lógica relacionada con la autenticación de usuarios. La autenticación consta de dos procesos principales: el inicio de sesión (signin) y el registro de usuarios (signup).

- AuthController: Es el encargado de gestionar las solicitudes del frontend y delegar el procesamiento de la lógica de negocio al AuthService. Este último se encarga de la validación de credenciales, creación de usuarios y emisión de tokens JWT para permitir la autenticación de las solicitudes futuras.
    - signinDto: Recibe las credenciales del usuario (como el correo electrónico y la contraseña) para validar su identidad.
    - signinDto: Recibe las credenciales del usuario (como el correo electrónico y la contraseña) para validar su identidad.

Componente	Descripción
signinDto	Recibe las credenciales de inicio de sesión (correo electrónico y contraseña). Validado por decoradores como @IsEmail() y @MinLength().
signupDto	Recibe los datos de registro (correo, contraseña, nombre). Incluye validación de correo y contraseña.
AuthController	Controlador que gestiona las solicitudes de inicio de sesión y registro, delegando en el servicio de autenticación.
AuthService	Servicio que maneja la lógica de negocio para la autenticación: validación de credenciales, registro de usuarios, y generación de tokens JWT.
Decoradores	@Body(), @Param(), @GetUser(): Usados para extraer datos de la solicitud o el contexto de ejecución.

Tabla 4.1: Responsabilidades de cada componente de la autorización en el BFF

Método	Descripción
signIn()	Verifica las credenciales del usuario, generando un JWT si son correctas.
signUp()	Crea un nuevo usuario tras verificar que no exista en la base de datos.
validateUser()	Verifica si las credenciales (usuario y contraseña) son correctas.
generateToken()	Genera un token JWT que se utilizará para autenticar futuras solicitudes.
Decoradores	@Body(), @Param(), @GetUser(): Usados para extraer datos de la solicitud o el contexto de ejecución.

Tabla 4.2: Métodos principales en el AuthService

El AuthController es el encargado de recibir las solicitudes HTTP del frontend. La interacción es la siguiente:

- En una solicitud de inicio de sesión (POST /auth/signin), el controlador recibe un signinDto, lo pasa al AuthService para validarlo y, si todo es correcto, genera un token JWT y lo devuelve al cliente.
- En una solicitud de registro de usuario (POST /auth/signup), el controlador recibe un signupDto, que es procesado por el AuthService para registrar al nuevo usuario, asegurándose de que no exista previamente. Luego, se genera y devuelve un JWT para la nueva cuenta.

Funcionalidad	**Descripción**
Validar credenciales de inicio de sesión	Verifica si las credenciales del usuario son correctas.
Crear un nuevo usuario	Registra un nuevo usuario en la base de datos.
Generar token JWT	Genera un token JWT para autenticación posterior.
Verificación de usuario existente	Verifica si un usuario con el mismo correo electrónico ya existe en la base de datos.
Decoradores	@Body(), @Param(), @GetUser(): Usados para extraer datos de la solicitud o el contexto de ejecución.

Tabla 4.3: Tabla Resumen de Funcionalidades de AuthService

La arquitectura de autenticación con DTOs, Controladores, y el Servicio de Autenticación proporciona una estructura clara y escalable para gestionar el inicio de sesión y registro de usuarios. Los decoradores y validaciones aseguran que los datos sean procesados correctamente antes de ser enviados a la lógica de negocio, y la integración con JWT permite manejar la autenticación de forma segura en todo el sistema.

Los decoradores auth.decorator, get-user.decorator y role-protected.decorator trabajan juntos para implementar un sistema de autenticación y autorización en el backend. Estos decoradores permiten manejar de manera sencilla la validación de tokens, la extracción de información del usuario y la gestión de permisos basados en roles, asegurando que solo los usuarios autenticados y autorizados puedan acceder a las funcionalidades del sistema.

Decorador	**Descripción**	Uso Principal
auth.decorator	Protege las rutas asegurando que solo los usuarios autenticados puedan acceder.	Se utiliza para proteger las rutas que requieren un token JWT válido.
get-user.decorator	Extrae la información del usuario autenticado (como ID o correo) del token JWT y lo pasa al controlador.	Se usa para obtener los datos del usuario dentro del controlador, facilitando la gestión de la sesión.
role-protected.decorator	Restringe el acceso a las rutas en función de los roles del usuario.	Se utiliza para restringir rutas a usuarios con roles específicos, como ”admin”, ”user”, etc.

Tabla 4.4: Tabla Resumen de Funcionalidades de Decoradores

Los guards son una parte crucial del sistema de autenticación y autorización, ya que controlan el acceso a las rutas en función de ciertos criterios, como la validez del token JWT y el rol del usuario. A continuación se detallan sus responsabilidades y funcionamiento.

El jwtAuth.guard es un guardia utilizado para proteger las rutas asegurando que solo los usuarios autenticados puedan acceder a ellas. Este guardia verifica que la solicitud tenga un token JWT válido y que este token contenga la información necesaria para autenticar al usuario.

- Responsabilidades:
  - Verificación del Token JWT: Asegura que la solicitud contenga un token JWT válido en el encabezado de la autorización (Authorization).
  - Autenticación del Usuario: Si el token es válido, el guardia permite que la solicitud continúe y pasa el control al siguiente componente. Si el token es inválido o no está presente, el acceso es denegado.
  - Configuración del Usuario Autenticado: Una vez que el token es validado, se coloca la información del usuario en el contexto de la solicitud, lo que permite su uso en otros componentes del sistema, como controladores y servicios.
- Funcionamiento:
  - El guardia actúa como un filtro en las rutas, interceptando las solicitudes entrantes.
  - Primero, extrae el token JWT de los encabezados HTTP.
  - Luego, utiliza el servicio de autenticación (por ejemplo, el AuthService) para verificar la validez del token.
  - Si el token es válido, el guardia permite que la solicitud continúe y pasa el control al controlador o servicio correspondiente.
  - Si el token no es válido o no está presente, se lanza una excepción UnauthorizedException, denegando el acceso.

El user-role.guard es un guardia utilizado para restringir el acceso a las rutas en función de los roles del usuario. Este guardia verifica si el usuario tiene el rol adecuado para ejecutar una acción específica, lo que permite implementar un control de acceso basado en roles.

- Responsabilidades:
  - Verificación de Roles: El guardia verifica si el usuario tiene el rol necesario para acceder a una ruta específica. Por ejemplo, podría verificar si el usuario es un ”admin” o un ”user”.
  - Autorización del Usuario: Si el usuario tiene el rol adecuado, el guardia permite que la solicitud continúe. Si el usuario no tiene el rol adecuado, se lanza una excepción ForbiddenException, bloqueando el acceso.

- Funcionamiento:
  - El guardia se aplica después de la autenticación. Una vez que el usuario ha sido autenticado mediante el JwtAuthGuard, el user-role.guard verifica si el usuario tiene el rol adecuado.
  - El rol del usuario suele estar presente en el token JWT, que se puede extraer y comparar con los roles requeridos para la ruta.
  - Si el usuario tiene el rol adecuado, el guardia permite que la solicitud continúe. Si no, se lanza una excepción ForbiddenException, indicando que el usuario no tiene permiso para acceder a esa ruta.

Guardia	**Descripción**
JwtAuthGuard	Verifica que la solicitud contenga un token JWT válido y autentica al usuario.
UserRoleGuard	Verifica que el usuario tenga el rol adecuado para acceder a una ruta específica.
role-protected.decorator	Restringe el acceso a las rutas en función de los roles del usuario.

Tabla 4.5: Tabla Resumen de Guards

Los guards jwtAuth.guard y user-role.guard son fundamentales en el proceso de autenticación y autorización de una aplicación. El jwtAuth.guard asegura que solo los usuarios autenticados puedan acceder a las rutas, validando los tokens JWT. Por otro lado, el user-role.guard garantiza que solo los usuarios con roles específicos puedan acceder a ciertas rutas, proporcionando un control granular sobre quién puede hacer qué dentro de la aplicación. Estos guards trabajan de forma conjunta para garantizar que tanto la autenticación como la autorización sean gestionadas adecuadamente, protegiendo así las rutas sensibles de la aplicación.

ValidRoles es un conjunto de constantes que se utilizan para definir los roles válidos en la aplicación. Este conjunto ayuda a garantizar que solo los usuarios con ciertos roles puedan acceder a rutas específicas.

- Responsabilidades:
  - Definición de Roles: Define los roles que los usuarios pueden tener, como ”admin”, ”user” y ”superuser”.
  - Reutilización: Proporciona una forma centralizada de definir y verificar roles en toda la aplicación.
  - Validación de Roles: Se utiliza para validar que el rol de un usuario coincida con los roles permitidos para acceder a determinadas rutas o funcionalidades

El JWT-Payload es la parte del token JWT que contiene la información del usuario y otros datos relevantes. Este payload se utiliza para identificar y autenticar al usuario cuando se presenta el token.

- Responsabilidades:
  - Almacenar Información del Usuario: Contiene datos clave del usuario, como su ID, email, roles y otras propiedades necesarias para la autenticación.
  - Duración del Token: También puede contener información sobre la expiración del token y otros metadatos relacionados con su validez.

Componente	**Descripción**	Ejemplo de Uso
JWTStrategy	Estrategia que valida y decodifica el token JWT, extrayendo la información del usuario y validando su autenticidad.	Se utiliza en el guardia JwtAuthGuard para autenticar solicitudes.
ValidRoles	Enum que define los roles válidos para los usuarios, como ”admin”, ”user”	Se usa para controlar el acceso a rutas según el rol del usuario.
JWT-Payload	Interfaz que describe el contenido del payload de un token JWT, que contiene datos sobre el usuario.	Se utiliza para representar los datos del usuario dentro del token JWT.

Tabla 4.6: Tabla Resumen de Componentes

El JWTStrategy, ValidRoles, y el JWT-Payload son componentes esenciales para manejar la autenticación y autorización de usuarios en una aplicación. El JWTStrategy permite verificar y validar los tokens JWT, el ValidRoles proporciona un sistema centralizado de roles para controlar el acceso a las rutas, y el JWT-Payload contiene la información clave del usuario. Juntos, estos elementos permiten crear un sistema robusto de control de acceso y seguridad para la aplicación.

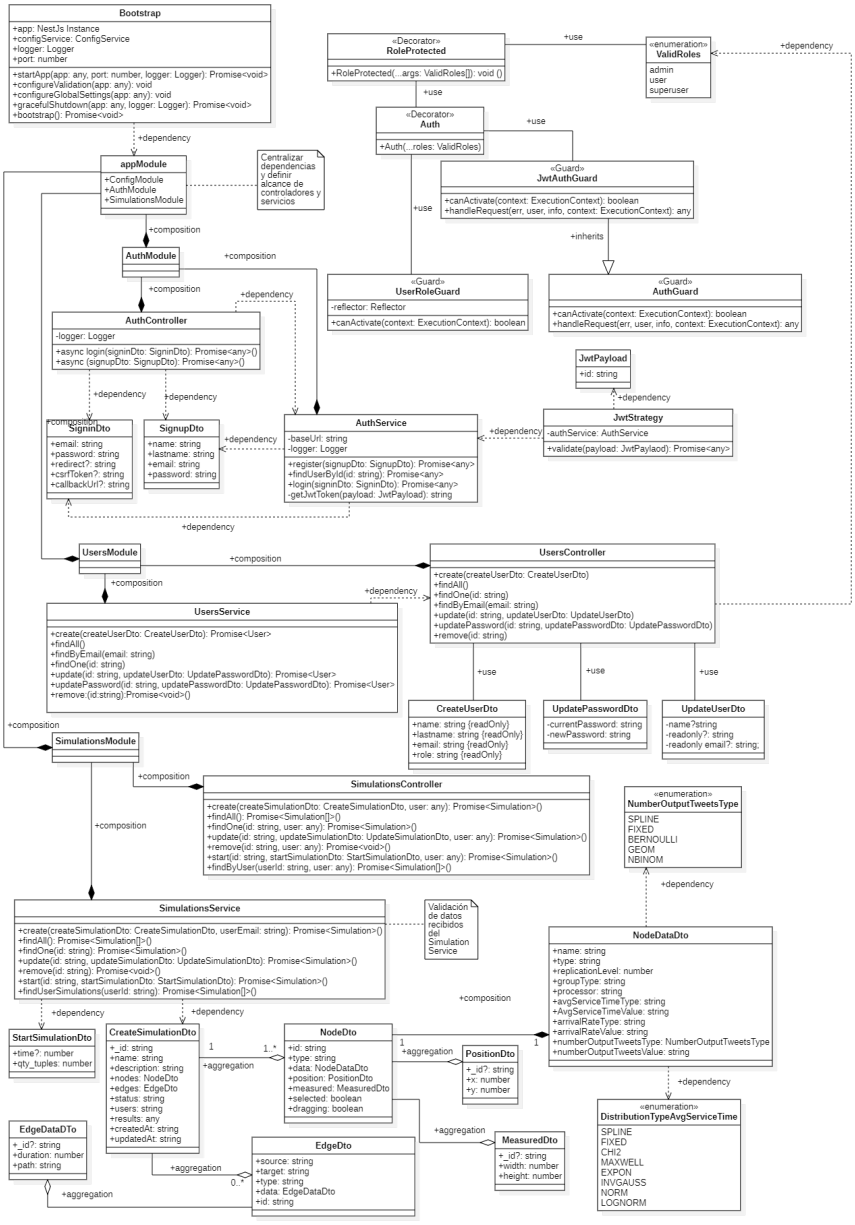

Figura 4.4: Diagrama de clases del BFF(Backend For Frontend)

User Service

El User Service es un componente clave dentro de la arquitectura de la aplicación, encargado de gestionar la lógica de negocio relacionada con los usuarios. Este servicio interactúa con la base de datos para realizar operaciones CRUD (Crear, Leer, Actualizar,

Eliminar) sobre los datos de los usuarios. Además, se encarga de manejar la validación de datos, el cifrado de contraseñas y la actualización de información sensible, como la contraseña del usuario. El diagrama de clases (ver Figura 4.5) a continuación describe las relaciones y responsabilidades de las distintas entidades involucradas en el User Service, mostrando cómo interactúan el UsersService, el UserController, y las entidades de datos, tales como User, CreateUserDto, UpdateUserDto, y UpdatePasswordDto. A través de estas clases, el User Service proporciona una estructura clara y modular para gestionar las funcionalidades relacionadas con los usuarios en el sistema.

- UsersService: Clase responsable de gestionar las operaciones CRUD (Crear, Leer, Actualizar, Eliminar) para los usuarios, utilizando el modelo User de MongoDB. Además de incluir funciones adicionales como la actualización de contraseñas, este servicio maneja excepciones específicas para cada caso, utilizando clases como BadRequestException, NotFoundException e InternalServerErrorException para manejar errores.

Método	Descripción	Excepciones
create(createUserDto: CreateUserDto)	Crea un nuevo usuario. Verifica que el correo no esté registrado previamente y hashea la contraseña antes de guardar el usuario	Lanza BadRequestException si el correo ya está registrado
finAll()	Recupera todos los usuarios almacenados en la base de datos MongoDB	
findOne(id:string)	Busca un usuario por su ID. Si no se encuentra o el ID es inválido, lanza excepciones específicas.	Lanza NotFoundException Si el usuario no existe.
findByEmail(email: string)	Busca un usuario por su correo electrónico Asegura que la contraseña sea incluida en la consulta.	Lanza NotFoundException Si el correo no corresponde a ningún usuario.
updatePassword(id: string, updatePasswordDto:UpdatePasswordDto)	Permite al usuario cambiar su contraseña, asegurando que la contraseña actual es correcta y la nueva cumple con los requisitos.	Lanza BadRequestException Si la contraseña actual no coincide o la nueva contraseña no es válida.
remove(id: string)	Elimina un usuario por su ID. Si el ID no corresponde a ningún usuario, lanza una excepción.	Lanza NotFoundException Si el usuario no existe.

Tabla 4.7: Descripción de funcionalidades de la clase UsersService

- UsersController: Responsable de manejar las solicitudes HTTP relacionadas con la gestión de usuarios. Esta clase expone los endpoints necesarios para interactuar con los servicios de usuario, utilizando el patrón REST y el servicio UsersService, el UsersController recibe las peticiones del BFF, las valida y la pasa al servicio correspondiente para que ejecuten las operaciones de negocio necesarias.

A continuación, presentamos la tabla correspondiente a la definición de endpoints presentes por el UsersController:

Método HTTP	Endpoint	Descripción
POST	/users	Crea un nuevo usuario en el sistema.
GET	/users	Recupera todoslos usuarios del sistema.
GET	/users/by-email/:email	Recupera un usuario por su correo electrónico.
GET	/users/:id	Recupera un usuario por su ID.
PATCH	/users/:id	Actualiza los datos de un usuario existente.
PUT	/users/:id/password	Actualiza la contraseña de un usuario existente.
DELETE	/users/:id	Elimina un usuario del sistema.

Tabla 4.8: Tabla resumen de Endpoints de UsersController

- UpdatePasswordDto: Objeto de transferencia de datos (DTO) utilizado para actualizar la contraseña de un usuario. Contiene los siguientes campos:
  - currentPassword: La contraseña actual del usuario (necesaria para verifica que la solicitud proviene del usuario correcto).
  - newPassword: La nueva contraseña que el usuario desea establecer (debe cumplir con los requisitos de seguridad, como una longitud mínima).
- UpdateUserDto: Es un DTO utilizado para actualizar los datos generales de un usuario. Puede incluir cualquier campo relevante para modificar la información del usuario, como:
  - Campos típicos: Nombre, correo electrónico, dirección, etc.
- User: Clase responsable de representar a un usuario en el sistema y define la estructura del documento en la base de datos de MongoDB. Contiene los siguientes campos:
  - _id: Identificador único del usuario en la base de datos (generado automáticamente por MongoDB).
  - email: El correo electrónico del usuario, que debe ser único.
  - password: Contraseña hash del usuario, que se almacena de manera segura usando la libreria bcrypt.
  - createdAt y updatedAt: Tiempos de creación y actualización del usuario.

- Bootstrap: Hace referencia al proceso en el que se inicializa el servicio y se configuran los módulos necesarios para su funcionamiento. Se invoca al método NestFactory.create() para crear la instancia de la aplicación y ejecutar el método app.listen() para iniciar el servidor.

- appModule: Módulo raíz responsable de encapsular toda la configuración del servicio, incluyendo otro módulos. Aquí se importan los módulos necesarios para la funcionalidad de la aplicación, como el UsersModule, MongooseModule(para la conexión a MongoDB).

- UsersModule: Módulo encargado de gestionar toda la lógica relacionada con los usuarios, incluyendo operaciones CRUD sobre los usuarios del sistema. Configura el UsersController y el UsersService.

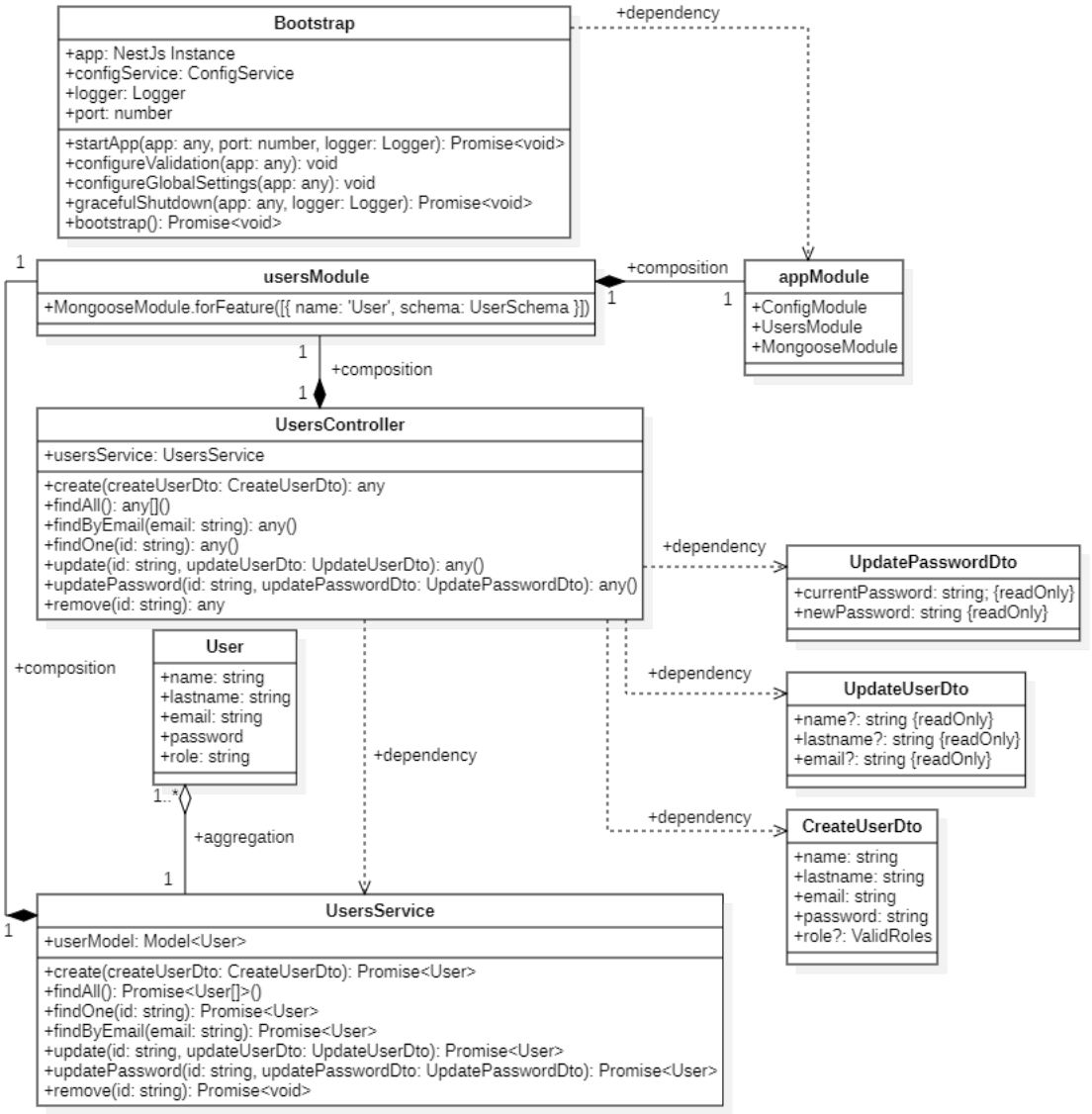

Figura 4.5: Diagrama de clases del User Service

Simulation Service

El Simulation Service es un componente central en la arquitectura de la aplicación, responsable de gestionar la lógica de negocio relacionada con las simulaciones. Este servicio interactúa con la base de datos para realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) sobre los datos de las simulaciones. Además, maneja la ejecución de los procesos de simulación, el almacenamiento de resultados y la actualización de los estados de las simulaciones. El diagrama de clases (ver Figura 4.6) a continuación ilustra las relaciones y responsabilidades de las distintas entidades involucradas en el Simulation Service, mostrando cómo interactúan el SimulationService, el SimulationController, y las entidades de datos como Simulation, CreateSimulationDto, UpdateSimulationDto, y SimulationResults. A través de estas clases, el Simulation Service ofrece una estructura organizada y escalable para gestionar las funcionalidades relacionadas con las simulaciones dentro del sistema, asegurando que el proceso sea eficiente y robusto.

- Bootstrap: Hace referencia al proceso en el que se inicializa el servicio y se configuran los módulos necesarios para su funcionamiento. Se invoca al método NestFactory.create() para crear la instancia de la aplicación y ejecutar el método app.listen() para iniciar el servidor.

- appModule: Módulo raíz responsable de encapsular toda la configuración del servicio, incluyendo otro módulos. Aquí se importan los módulos necesarios para la funcionalidad de la aplicación, como el SimulationsModule, MongooseModule(para la conexión a MongoDB) y ConfigModule para cargar las variables de entorno.

- SimulationsModule: Módulo encargado de agrupar todos los componentes necesarios para gestionar los modelos de simulación. Este módulo define la estructura y organización de los servicios, controladores y entidades relacionados con las simulaciones. Facilitando la inyección de dependencias, configuración de rutas y servicios necesarios para crear, actualizar y consultar modelos de simulación en el sistema.

- SimulationsService: : Clase responsable de gestionar la lógica de negocio relacionada con las simulaciones. Este servicio maneja las operaciones de creación, actualización, y consulta de simulaciones, interactuando con la base de datos. Además, maneja las excepciones relacionadas con las simulaciones, como errores de validación y problemas internos del servidor.

Método	Descripción	Excepciones
create(createSimulationDto: CreateSimulationDto)	Crea una nueva simulación. Verifica que los datos estén completos y guarda la simulación en la base de datos.	Lanza BadRequestException si los datos de la simulación son inválidos.
findAll(userId: string)	Recupera todas las simulaciones almacenadas en la base de datos.	
findOne(id: string)	Busca una simulación por su ID. Si no se encuentra o el ID es inválido, lanza excepciones específicas.	Lanza NotFoundException si la simulación no existe.
update(id: string, updateSimulationDto: UpdateSimulationDto)	Actualiza los datos de una simulación existente en la base de datos.	Lanza NotFoundException si la simulación no existe.
addResults(id: string, results: ResultsDto)	Añade resultados a una simulación existente.	Lanza NotFoundException si la simulación no existe.
remove(id: string)	Elimina una simulación por su ID. Si el ID no corresponde a ninguna simulación,lanza una excepción.	Lanza NotFoundException si la simulación no existe.

Tabla 4.9: Descripción de funcionalidades de la clase SimulationsService

- SimulationsController: Responsable de gestionar las peticiones HTTP relacionadas con las simulaciones. Actúa como la capa que recibe las solicitudes del cliente (por ejemplo, del frontend o de otros servicios), las procesa a través del SimulationService y devuelve una respuesta adecuada. Este controlador incluye los endpoints necesarios para la creación, consulta, actualización y eliminación de simulaciones, así como para la adición de resultados a las simulaciones existentes.

A continuación, presentamos la tabla correspondiente a la definición de endpoints presentes por el SimulationsController:

Método	Endpoint	Descripción
POST	/simulations	Crea una nueva simulación.
GET	/simulations	Recupera todas las simulaciones almacenadas.
GET	/simulations/:id	Recupera una simulación por su ID.
PATCH	/simulations/:id	Actualiza los datos de una simulación existente.
PATCH	/simulations/:id/results	Agrega los resultados a una simulación.
DELETE	/simulations/:id	Elimina una simulación por su ID.

Tabla 4.10: Tabla resumen de Endpoints de SimulationsController

- Simulations: Representa el modelo de una simulación dentro del sistema. Responsable de la validación de la estructura de estos modelos, contiene información fundamental como su identificador único (_id), el usuario que la creó (user), su nombre

(name), el estado actual (status), las fechas de creación y actualización (createdAt y updatedAt), y la información critica de los nodos y sus bordes (edges). Esta clase es fundamental para gestionar el ciclo de vida de una simulación, desde su creación hasta la recopilación de resultados.

Propiedad	Tipo	Descripción
_id	string	Identificador único del modelo de simulación.
name	string	Nombre del modelo de simulación.
description	string	Descripción del modelo de simulación
nodes	NodeDto	Fecha de inicio de la simulación.
edges	EdgeDataDto	Fecha de finalización de la simulación.
status	SimulationStatus	Estado de la simulación (e.g., ”in-progress”, ”completed”)
users	string	Identificador único del usuario
results	any	Datos de resultados del modelo de simulación
createdAt	string	Fecha de creación del modelo de simulación
updatedAt	string	Fecha de actualización del modelo de simulación

Tabla 4.11: Descripción de la clase Simulation

- NodeDto: Clase responsable de representar de las entidades nodos dentro del modelo de simulación.
  - _id: Es un identificador único para cada nodo. Este valor es crucial para identificar de manera única cada nodo en el flujo, lo cual es importante para las operaciones de actualización, eliminación o referencia entre nodos.
  - type: Indica el tipo de nodo que fue utilizado para representarlo en el espacio 2d.
  - data: Contiene los datos esenciales asociados con el nodo, utiles para ser utilizados durante el proceso de simulación como el nombre del nodo, el tipo de nodo (Spout o Bolt), nivel de replicación, agrupamiento, procesador, tiempo promedio de servicio, tasa de arribo y número de Tweets de salida.
  - position: Representa la posición del nodo en el espacio 2D. En React Flow, la posición de cada nodo es una coordenada X, Y que define dónde se coloca en el área de visualización. Esta propiedad es fundamental para renderizar el nodo correctamente en la interfaz.
  - measures: Generalmente se usa para almacenar detalles adicionales sobre el nodo, como sus dimensiones (ancho, alto, etc.). Esto puede ser útil si necesitas ajustar la apariencia de los nodos dependiendo de su tamaño en el espacio de trabajo.

  - selected: Indica si el nodo está seleccionado en la interfaz. En React Flow, la selección de nodos permite aplicar estilos específicos, como cambiar el borde o mostrar detalles adicionales cuando el usuario interactúa con ellos.
  - dragging: Indica si el nodo está siendo arrastrado por el usuario. Esta propiedad puede usarse para manejar interacciones y animaciones cuando el nodo está siendo movido dentro de la visualización, lo que es una característica común en aplicaciones que permiten la manipulación interactiva de los diagramas.

- EdgeDataDto: Representa los bordes (edges) en el modelo de simulación y se encarga de gestionar las conexiones entre los diferentes nodos del modelo de simulación. Un edge conecta 2 nodos, permitiendo el flujo de información o recursos entre ellos. Cada edge almacena información como su identificador, el nodo de origen y destino, y parámetros adicionales.

Las clases NodeDataDTO y EdgeDataDTO son esenciales en el proceso de simulación, ya que permiten estructurar y gestionar de manera eficiente los datos de los nodos y las conexiones (aristas) entre ellos, lo cual es fundamental para modelar y ejecutar la simulación. Durante el proceso de simulación es esencial calcular como las aristas afectan la propagación de flujos de datos y como las características de cada borde impactan en el resultado final de la simulación. A medida que los nodos interactúan entre sí a través de sus conexiones, los resultados de la simulación pueden variar directamente de como están relacionados todos los nodos de un modelo de simulación.

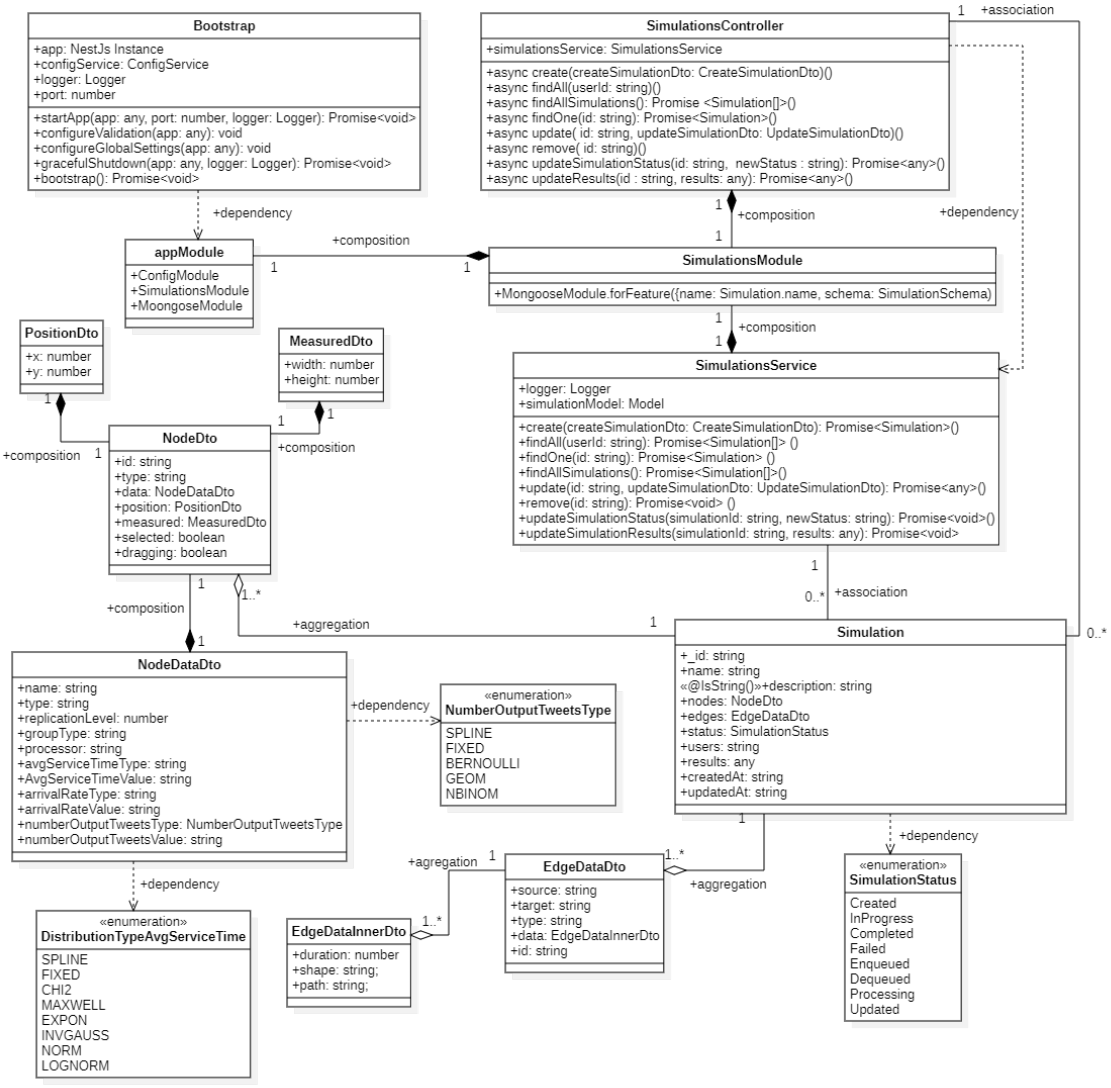

Figura 4.6: Diagrama de clases del Simulation Service

Producer Service

El Producer Service gestiona la validación y el envío de datos de simulación a las colas de RabbitMQ, facilitando un flujo desacoplado entre los servicios. Actúa como puente entre los datos de entrada de los modelos de simulación y su procesamiento por el Consumer Service. Integrado con el RabbitMQ Module, configura y mantiene las conexiones necesarias para garantizar una transmisión confiable de los mensajes. Utiliza modelos de datos como NodeDataDto y EdgeDataDto para estructurar adecuadamente la información enviada.

- Bootstrap: Hace referencia al proceso en el que se inicializa el servicio y se configuran los módulos necesarios para su funcionamiento. Se invoca al método NestFactory.create() para crear la instancia de la aplicación y ejecutar el método app.listen() para iniciar el servidor.
- appModule: Módulo raíz responsable de encapsular toda la configuración del servicio, incluyendo otro módulos. Aquí se importan los módulos necesarios para la funcionalidad de la aplicación, como el ProducerModule, RabbitMQModule y ConfigModule para cargar las variables de entorno.
- Producer Module: Módulo encargado de organizar e integrar todos los componentes relacionados con el funcionamiento del Producer Service. Este módulo encapsula la configuración del servicio, incluyendo la conexión con RabbitMQ a través del RabbitMQModule, y expone el ProducerService como el punto principal para manejar la publicación de mensajes en las colas. Agrupa dependencias clave como los DTOs y modelos necesarios para procesar los datos de los modelo de simulación, asegurando una configuración modular, escalable y reutilizable dentro de la arquitectura del sistema
- RabbitMQModule: Módulo especializado que encapsula toda la configuración y funcionalidad necesaria para interactuar con RabbitMQ. Este módulo abstrae detalles técnicos, como la conexión, autenticación y configuración de colas, proporcionando una interfaz simplificada para los servicios que lo utilizan, como el ProducerService. Dentro del módulo se define la configuración del cliente, como las credenciales, el host y los intercambios, además de manejar el reintento de conexiones y la recuperación ante fallos. Esto garantiza una comunicación eficiente y confiable entre los servicios mediante colas, lo que resulta esencial para el procesamiento asíncrono y desacoplado del sistema.
- ProducerController: Capa de control dentro del ProducerService, diseñada para manejar las solicitudes HTTP relacionadas con la gestión y encolamiento de simulaciones. Este controlador recibe datos validados desde el cliente que pasan por el BFF, procesa y delega las operaciones al ProducerService.

A continuación, se presenta una tabla resumen con los endpoints definidos en el ProducerController:

Método HTTP	Endpoint	Descripción	Parámetros	Excepciones
POST	/:id/start	Inicia el proceso de simulación encolando los datos de una simulación creada.	- id (Path): ID de la simulación. - simulationData (Body): Datos validados del modelo de simulación (ValidatedSimulationDto). -startSimulationDto (Body): Datos adicionales para iniciar la simulación (SimulationStartDto)	-BadRequestException: Datos de entrada inválidos. -HttpException: Errores en el proceso de encolamiento o conexión con RabbitMQ

Tabla 4.12: Tabla resumen de Endpoints de ProducerController

- ProducerService: Encargado de orquestar el inicio de simulaciones en el sistema. Su principal responsabilidad es procesar los datos validados de simulaciones, actualizar su estado en el servicio correspondiente y publicar los datos en una cola de RabbitMQ para ser consumidos por otros servicios. Además, este servicio asegura la limpieza y unificación de los datos, manejando errores en cada etapa del flujo para garantizar la integridad del proceso. A continuación, se detallan los principales métodos que forman parte de este servicio:

Método HTTP	Descripción
startSimulation	Valida los datos de entrada, actualiza el estado a ”InProgress”, Limpia los datos de nodos y edges, publica los datos a RabbitMQ para iniciar la simulación.
cleanAndMergeData	Limpia y formatea la estructura de nodos y edges, incluyendo atributos específicos según el tipo (B o S), para asegurar consistencia y simplicidad.
publishToRabbitMQ	Publica un mensaje en RabbitMQ. Si ocurre un error, lanza una excepción RPC detallada.
updateSimulationStatus	Actualiza el estado de la simulación en el servicio de simulaciones usando un endpoint REST.

Tabla 4.13: Descripción de métodos existentes en el ProducerService

- RabbitMQService: Responsable de manejar la comunicación con el servidor de RabbitMQ. Este servicio utiliza RabbitMQConfig para obtener las configuraciones necesarias, como la URL de conexión, las opciones de las colas y los parámetros de la Dead Letter Queue (DLQ). Proporciona funcionalidades clave para establecer y mantener la conexión con RabbitMQ, gestionar colas y publicar mensajes, además

de procesar mensajes fallidos almacenados en la DLQ. Se asegura de manejar eventos de error y reconexión de manera automática para garantizar la disponibilidad continua del sistema.

Método HTTP	Descripción
onModuleInit	Inicializa la conexión con RabbitMQ al iniciar el módulo.
onModuleDestroy	Detiene la verificación de la conexión y cierra la conexión y el canal al destruir el módulo.
initializeRabbitMQConnection	Configura la conexión con RabbitMQ Establece el canal, declara la cola principal y configura la DLQ.
disconnectRabbitMQ	Cierra el canal y la conexión con RabbitMQ, limpiando los recursos.
publishMessage	Publica un mensaje en la cola principal de RabbitMQ y actualiza el estado de la simulación a ”Enqueued”.
consumeDLQOnStart	Configura el consumo de mensajes de la DLQ al iniciar el servicio.
processFailedMessage	Procesa mensajes fallidos de la DLQ y actualiza su estado a ”Failed” en el sistema.
handleConnectionClose	Maneja el cierre inesperado de la conexión con RabbitMQ, intentando reconectarse.
handleConnectionError	Maneja errores en la conexión con RabbitMQ y activa un intento de reconexión.
reconnectWithDelay	Intenta reconectar con RabbitMQ después de un retraso configurable.
startConnectionCheck	Inicia una verificación periódica de la conexión con RabbitMQ.
stopConnectionCheck	Detiene la verificación periódica de la conexión.

Tabla 4.14: Descripción de métodos existentes en el RabbitMQService

La clase StartSimulationDTO contiene valores esenciales como qty_tuples y opcionales como time, que se utilizan junto con los datos limpios del modelo de simulación para enviarlos a la cola de RabbitMQ. Estos valores permiten que el proceso de simulación sea configurado y gestionado correctamente en el flujo de mensajes entre servicios. El DTO proporciona la estructura necesaria para que el RabbitMQService publique los mensajes de manera adecuada, asegurando que el ProducerService reciba toda la información necesaria para iniciar la simulación.

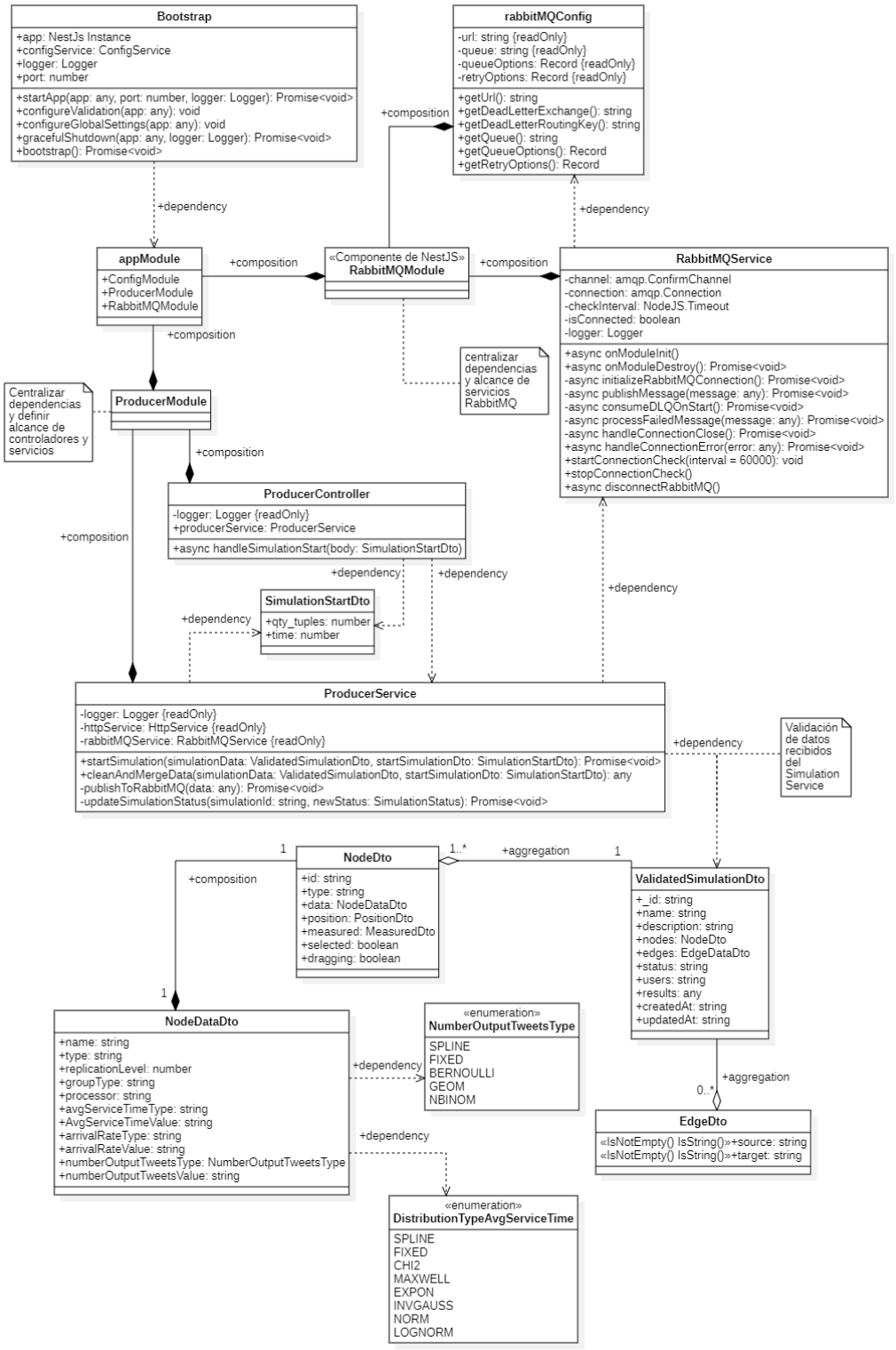

Figura 4.7: Diagrama de clases del Producer Service

Consumer Service

El Consumer Service es responsable de consumir los mensajes encolados en RabbitMQ y procesar los datos de simulación recibidos. Actúa como el componente encargado de ejecutar las simulaciones, actualizar el estado de las mismas y almacenar los resultados en la base de datos. Integrado con el RabbitMQ Module, establece las conexiones necesarias para recibir de manera confiable los mensajes del Producer Service. Utiliza servicios como el Processing Service para gestionar la lógica de simulación y el HttpClient Service para interactuar con el Simulation Service y actualizar los resultados.

- Bootstrap: Hace referencia al proceso en el que se inicializa el servicio y se configuran los módulos necesarios para su funcionamiento. Se invoca al método NestFactory.create() para crear la instancia de la aplicación y ejecutar el método app.listen() para iniciar el servidor.

- appModule: Módulo raíz responsable de encapsular toda la configuración del servicio, incluyendo otro módulos. Aquí se importan los módulos necesarios para la funcionalidad de la aplicación, como el ProcessingModule, RabbitMQModule, HttpClientModule y el ConfigModule para cargar las variables de entorno.

- Processing Module: Módulo encargado de gestionar la lógica de procesamiento de modelos de simulación recibido desde RabbitMQ. Su función principal es recibir los datos de entrada, ejecutar los procesos de simulación correspondientes y preparar los resultados para ser enviados al Simulation Service. Este módulo organiza las diferentes tareas necesarias para completar el ciclo de simulación de manera eficiente, delegando responsabilidades especificas a otros microservicios, como el HttpClientService para enviar una solicitud HTTP al Simulation service actualizando el estado y los resultados del modelo de la simulación. Además, actúa como un punto de integración entre el flujo de mensajes y el procesamiento de la lógica de negocio.

- RabbitMQModule: Módulo responsable de gestionar la conexión y comunicación con RabbitMQ, donde se encolan los mensajes enviados por parte del Producer Service. Este módulo maneja la suscripción a las colas de mensajes y asegura que los mensajes sean recibidos de manera confiable para ser procesados. A través de este módulo, El Consumer Service escuchando de forma continua las colas y, cuando llega un nuevo mensaje, lo pasa al ProcessingModule para su procesamiento. Además, gestiona la configuración de las conexiones, como las credenciales de acceso y los parámetros de la cola, garantizando una interacción estable y eficiente con RabbitMQ.

- HttpClientModule: Módulo encargado de gestionar la comunicación HTTP entre el Consumer Service y el Simulation Service. Su principal responsabilidad es realizar

peticiones HTTP para actualizar el estado de los modelo de simulación y resultados obtenidos del proceso de simulación en la base de datos de MongoDB. Este módulo permite al Consumer Service interactuar con el Simulation Service, enviando y recibiendo datos necesarios para mantener la coherencia de la simulación en todo el sistema. Utiliza servicios como el HttpClientService para realizar solicitudes RESTful, asegurando que la información se actualice correctamente en el Simulation Service y se refleje en la base de datos.

- Processing Service: Encargado de manejar la lógica central de procesamiento de los mensajes de simulación. Su principal función es recibir los mensajes consumidos desde RabbitMQ, ejecutar la simulación correspondiente utilizando Simstream y preparar los resultados para su almacenamiento o actualización en la base de datos. Este servicio organiza y coordina las tareas necesarias para procesar los datos de entrada, como la ejecución de cálculos, la manipulación de estructuras de datos y la integración con otros módulos como el HttpClientService para actualizar el estado de la simulación en el Simulation Service. El ProcessingService también asegura que los resultados sean generados de manera eficiente y se manejen adecuadamente, delegando tareas específicas a otros servicios cuando sea necesario.

Método	Descripción	Parámetros
processMessage	Maneja los mensajes recibidos de RabbitMQ, valida los datos, crea los archivos necesarios para la simulación, ejecuta la simulación y actualiza el estado de la simulación.	message: ValidatedSimulationDto
createTopologyFile	Crea el archivo topology.dat con las relaciones entre nodos.	data: any
createNodesDictionary	Crea un diccionario de nodos y una lista de relaciones entre ellos.	data: any
createNodesDetailFile	Crea el archivo nodes_detail.dat con los detalles de los nodos de la simulación.	data: any
createNodesDetails	Crea un arreglo con los detalles de los nodos, dependiendo de su tipo (S o B).	data: any
runSimulation	Ejecuta la simulación usando los archivos creados previamente y retorna la ruta del archivo de salida.	topologyFilePath: string, nodeDetailsFilePath: string, qty_tuples: number, time?: number
executeCommand	Ejecuta un comando en el sistema operativo usando exec y devuelve una promesa.	command: string
verifyFilesNotEmpty	Verifica que los archivos proporcionados no estén vacíos.	filePaths: string[]

Tabla 4.15: Descripción de métodos existentes en el ProcessingService

- RabbitMQService: Encargado de manejar la lógica de interacción con RabbitMQ. Este servicio se asegura de establecer y mantener la conexión con el servidor RabbitMQ, gestionando la suscripción a las colas específicas y garantizando la recepción

de los mensajes en tiempo real. Además, es responsable de configurar las colas, intercambios y enrutamientos necesarios para que los mensajes fluyan correctamente desde el Producer Service hacia el Consumer Service. El RabbitMQService también maneja los aspectos relacionados con el procesamiento asíncrono, asegurando que los mensajes sean consumidos de manera eficiente y entregados al ProcessingService para su posterior tratamiento.

A continuación, se presenta una tabla resumen de los métodos presentes en RabbitMQService:

Método	Descripción
onModuleInit()	Se ejecuta cuando el módulo se inicializa. Intenta establecer la conexión con RabbitMQ y comienza el chequeo periódico de la conexión.
onModuleDestroy()	Se ejecuta cuando el módulo se destruye. Detiene el chequeo periódico y cierra la conexión con RabbitMQ.
onApplicationShutdown()	Se ejecuta cuando la aplicación está siendo apagada. Cierra la conexión con RabbitMQ.
initializeRabbitMQConnection()	Inicializa la conexión a RabbitMQ y crea el canal de comunicación. Configura la prefetch para consumir un solo mensaje a la vez y comienza a consumir mensajes.
consumeMessages()	Consume los mensajes de la cola especificada, procesa los mensajes y envía un ACK a RabbitMQ para confirmar que el mensaje ha sido recibido correctamente.
checkConnection()	Verifica si la conexión y el canal a RabbitMQ están activos. Si no lo están, intenta reconectar.
startConnectionCheck()	Inicia un chequeo periódico de la conexión a RabbitMQ, que se realiza cada 30 segundos.
stopConnectionCheck()	Detiene el chequeo periódico de la conexión.
disconnectRabbitMQ()	Cierra el canal y la conexión a RabbitMQ de manera segura.
log	Un objeto de Logger utilizado para registrar información de depuración, advertencias y errores.
assertQueue(queueName: string)	Verifica la existencia de una cola. Si no existe, la crea con las configuraciones predeterminadas.
stopConnectionCheck	Detiene la verificación periódica de la conexión.

Tabla 4.16: Descripción de métodos existentes en el RabbitMQService

Propiedad	Descripción
connection	Almacena la conexión a RabbitMQ.
channel	Almacena el canal de comunicación con RabbitMQ.
checkInterval	Almacena el identificador de la función setInterval para el chequeo periódico de la conexión.
isReconnecting	Indicador de si el servicio está intentando reconectar.
isConnected	Indicador de si la conexión a RabbitMQ está activa.
isConsuming	Indicador de si el servicio está consumiendo mensajes de la cola.
logger	Un objeto Logger para registrar la información relevante durante la ejecución.

Tabla 4.17: Propiedades de RabbitMQ Service

- index: Clase contiene la función principal parseFile, la cual es responsable de procesar un archivo de salida línea por línea utilizando un flujo de lectura. Cada línea se pasa a la función dispatchLine para su procesamiento y transformación. El resultado se almacena en un objeto context que organiza los datos en varias categprías, como arrivalRates, numberTuples, serviceTimes, processors, nodes y stadistics. Esta clase utiliza los módulos de NodeJS fs (para operaciones de archivos) y readLine (para leer líneas de forma eficiente). En caso de errores, se registra un mensaje de error y se lanza una excepción.

ropiedad del Contexto	Descripción
arrivalRates	Lista de tasas de llegada procesadas del archivo.
numberTuples	Lista de tuplas numéricas extraídas del archivo.
serviceTimes	Lista de tiempos de servicio calculados a partir de las líneas del archivo.
processors	Lista de procesadores identificados en el archivo.
nodes	Lista de nodos configurados con base en la información del archivo.
statistics	Datos estadísticos derivados del procesamiento del archivo.
logger	Un objeto Logger para registrar la información relevantedurante la ejecución.

Tabla 4.18: Propiedades de la clase index

- parserfile: Clase que define una colección de patrones y funciones de manejo que procesan líneas específicas del archivo de salida generado por el simulador SimStream. La lógica principal se encuentra en la función dispatchLine, que compara cada línea con un conjunto de patrones regulares y, si hay coincidencia, ejecuta la función asociada con el patrón. Este archivo organiza y estructura el procesamiento de datos basándose en patrones definidos, como procesadores, tasas de llegada, tiempos de servicio, estadísticas y otros. También maneja casos donde una línea no coincide con ningún patrón, registrando una advertencia.

A continuación, se presentan los patrones utilizados para la lectura del archivo de salida generado por el simulador SimStream:

Patrón	Función Manejadora	Descripción
^\s*$	() =>{}	Maneja líneas vacías; no realiza ninguna acción.
/PROCESSOR: (\S+) - CORE: Core_(\d+) time_in_use: ... utilization: (-?\d+(\.\d+)?([eE][-+]?\d+)?)/	parseCore	Procesa información de núcleos del procesador.
/PROCESSOR: (\S+) in_use: (-?\d+(\.\d+)?([eE][-+]?\d+)?)/	parseProcessor	Extrae datos generales del procesador, como memoria usada y accesos acumulados.
/PROCESSOR utilization: (-?\d+(\.\d+)?([eE][-+]?\d+)?)/	parseProcessorUtilization	Procesa la utilización del procesador.
/NODE:\s*(\S+)_?(\d*)\s*use_time: ... tuples:\s*([-+]?\d+) replica:\s*(\d+)/	parseNodeReplica	Procesa información sobre réplicas y rendimiento de nodos.
/^\s*utilization:(-?\d+(\.\d+)?) throughput:(-?\d+(\.\d+)?) replicas:(\d+)/	parseNodeSummary	Procesa el resumen de rendimiento de un nodo.
/ARRIVAL_RATE (\d+(\.\d+)?)/	parseArrivalRate	Extrae tasas de llegada para los eventos del simulador.
/SERVICE_TIME (\S+) (\d+(\.\d+)?)/	parseServiceTime	Procesa el tiempo de servicio asociado a componentes específicos.
/NUMBER_OF_TUPLES (\S+) (\d+)/	parseNumberOfTuples	Extrae la cantidad de tuplas procesadas o generadas.
/tuples generated:(\d+) tuples processed:(\d+) ... total simulation time:(\S+)/	parseStadistics	Procesa las estadísticas generales de la simulación, como tiempo total y rendimiento.

Tabla 4.19: patrones utilizados para parserfile

- parseHandlers: Esta clase implementa un conjunto de funciones para analizar y procesar líneas de texto generadas por simulaciones o sistemas distribuidos. Estas funciones están diseñadas para extraer información específica sobre métricas clave, como tasas de llegada, tiempos de servicio, utilización de procesadores, tiempos de uso de nodos, y detalles de réplicas. Cada función utiliza expresiones regulares para identificar y capturar los datos relevantes de una línea de entrada y organiza estos datos en un objeto context estructurado. Este contexto actúa como un contenedor que almacena resultados agregados, lo que permite representar información compleja de manera jerárquica y fácilmente accesible. El código incluye validaciones y estrategias para evitar duplicados, manejar datos inválidos o faltantes, y asignar valores por defecto cuando sea necesario.

Método	Propósito	Entrada	Salida
parseArrivalRate	Extrae las tasas de llegada (ARRIVAL_RATE) y las agrega al contexto.	Línea de texto con formato: ARRIVAL_RATE <rate>	Agrega arrivalRate al contexto.
parseNumberOfTuples	Analiza el número de tuplas por nodo y las organiza por base y detalles.	Línea de texto con formato: NUMBER_OF_TUPLES <nodeName><number>	Agrega numberTuples al contexto.
parseServiceTime	Extrae los tiempos de servicio (SERVICE_TIME) para cada nodo y los organiza.	Línea de texto con formato: SERVICE_TIME <nodeName><time>	Agrega serviceTimes al contexto.
parseProcessor	Captura métricas generales de los procesadores, como uso promedio y memoria acumulativa.	Línea de texto con formato: PROCESSOR: <name>in_use: <value>- average memory=<value>- max memory=<value>– accs=<value>– cumm=<value>	Agrega procesadores al contexto.
parseCore	Extrae detalles de uso por núcleo dentro de un procesador específico.	Línea de texto con formato: PROCESSOR: <name>- CORE: Core_<id> time_in_use: <value> total_time: <value> utilization: <value>	Agrega núcleos a procesadores.
parseProcessorUtilization	Actualiza la utilización global del último procesador procesado.	Línea de texto con formato: PROCESSOR utilization: <value>	Actualiza la utilización del procesador.
parseNodeSummary	Captura métricas de resumen de un nodo, como utilización, rendimiento y réplicas.	Línea de texto con formato: utilization:<value> throughput:<value> replicas:<value>	Actualiza valores en el nodo.
parseNodeReplica	Analiza métricas detalladas de réplicas de nodos, incluyendo tiempos y utilización por réplica.	Línea de texto con formato: NODE: <name>use_time:<value> total_time:<value> utilization:<value> throughput:<value> avg_resp_time:<value> tuples:<value>	Agrega detalles a nodos.

Tabla 4.20: funciones utilizados para parseHandlers

- HttpClientService: Diseñada para gestionar la comunicación entre un cliente y el servicio de simulación, manejando actualizaciones del estado de las simulaciones y enviando resultados asociados. Este servicio utiliza el módulo HttpService de Axios en NestJS, con funciones como updateSimulationStatus para actualizar el estado de una simulación y sendSimulationResults para agregar resultados. Ambas funciones

incluyen validaciones estrictas de parámetros, construcción de URLs dinámicas basadas en variables de entorno, y manejo robusto de errores para garantizar confiabilidad. El servicio también implementa registros de operaciones (Logger) para rastrear solicitudes y depurar problemas eficientemente.

Método	Propósito	Entrada	Salida
updateSimulationStatus	Actualiza el estado de una simulación en el servicio de simulación.	simulationId: string, newStatus: SimulationStatus	Ninguna. Lanza excepciones en errores.
validateUpdateParameters	Valida los parámetros necesarios para actualizar el estado de una simulación.	simulationId: string, newStatus: SimulationStatus	Ninguna. Lanza excepciones en errores.
constructSimulationUrl	Construye la URL para la solicitud de actualización del estado de la simulación.	simulationId: string	URL como cadena de texto.
handleResponse	Procesa la respuesta del servicio para actualizaciones de estado exitosas o fallidas.	response: any, simulationId: string, newStatus: SimulationStatus	Ninguna. Lanza excepciones en errores.
handleError	Maneja errores ocurridos durante el intento de actualizar el estado de una simulación.	error: any, simulationId: string, newState: SimulationStatus	Ninguna. Lanza excepciones.
sendSimulationResults	Envía los resultados asociados a una simulación específica.	simulationId: string, newResults: any	Ninguna. Lanza excepciones en errores.

Tabla 4.21: funciones utilizados por HttpClientService

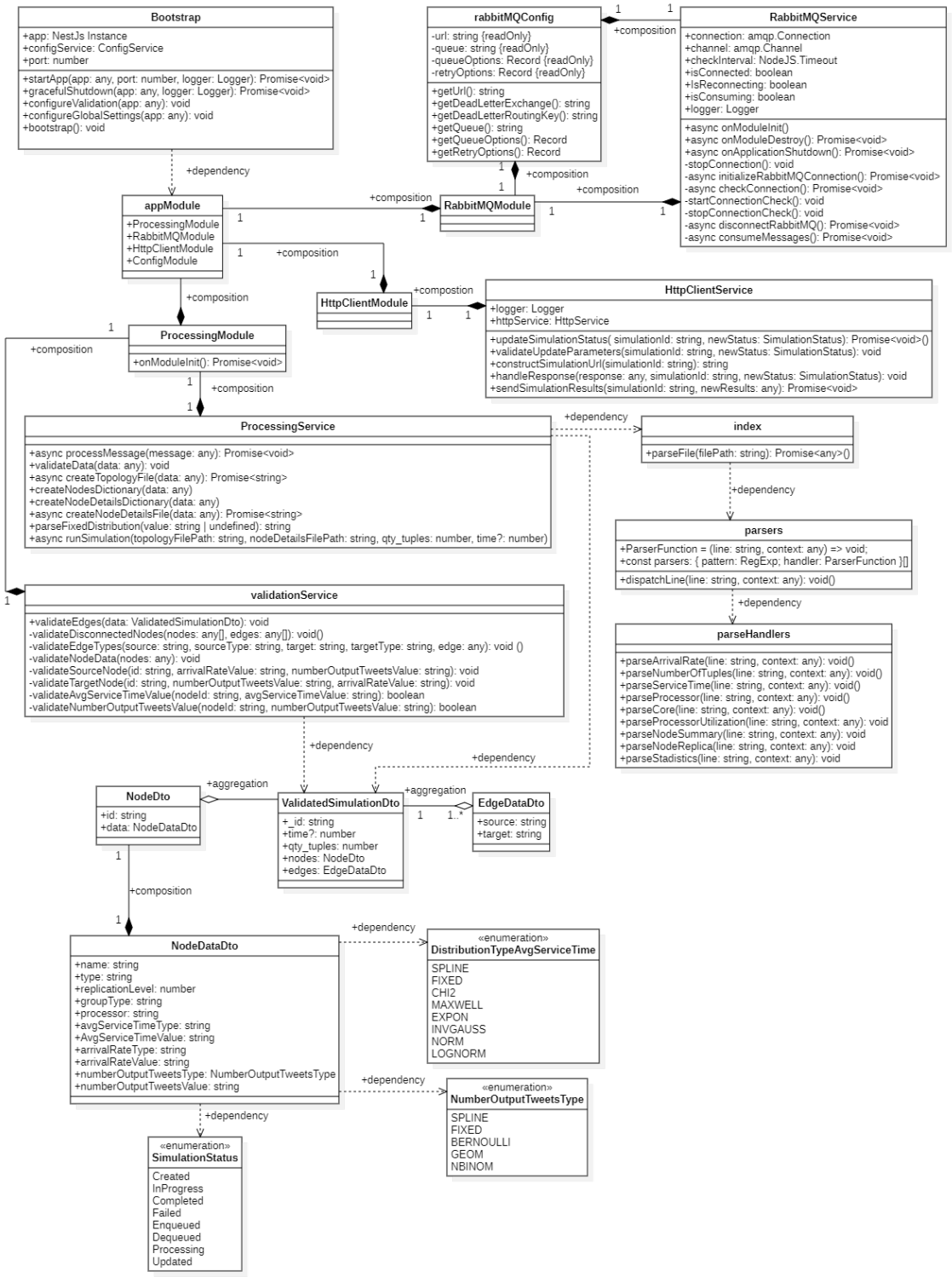

Figura 4.8: Diagrama de clases del Consumer Service

4.3. Diseño de Datos

4.3.1. Diccionario de Datos

En esta sección se presenta el diccionario de datos, el cual describe detalladamente las entidades y atributos claves utilizados por los diferentes servicios del sistema. El diccionario proporciona una referencia completa para entender como se estructuran y validan los datos en cada uno de los servicios involucrados, permitiendo asegurar la consistencia y claridad en la comunicación entre los componentes del sistema.

A continuación, se describen los diccionarios de datos para los servicios más relevantes del sistema, que incluyen los campos de entrada y salida de los servicios, así como las restricciones y valores permitidos por cada uno de los atributos.

Diccionario de Datos

A continuación, presentamos la estructura de la entidad *User*, que representa a un usuario en el sistema. Esta entidad contiene los campos esenciales para la gestión de usuarios, incluye el nombre, apellido, correo electrónico, contraseña y rol. Cada uno de estos campos es utilizado para autenticar y autorizar a los usuarios dentro del sistema. Presentamos los detalles de los campos incluidos en esta entidad:

- name: Representa el nombre del usuario. Este campo es obligatorio y debe ser un valor de tipo cadena.
- lastname: Representa el apellido del usuario. Este campo también es obligatorio y debe ser una cadena (string).
- email: Este campo representa el correo electrónico único del usuario, utilizado para la autenticación. Este campo es obligatorio, debe ser único y no se permite duplicación de correos electrónicos
- password: La contraseña del usuario, es de carácter obligatoria para la autenticación. Esta campo está marcado como select: false, lo que que significa que no se incluye en las consultas estándar, protegiendo la seguridad de la contraseña, además de esto se ha de recalcar que la contraseña es almacenada de forma encriptada en la base de datos de MongoDB, dándole otro nivel más de protección a la contraseña del usuario al momento de su creación.
- role: Define el rol del usuario en el sistema, por defecto al momento de crear un usuario se le asigna 'user' a este campo. Es de carácter obligatorio y se utiliza para definir si el usuario tiene privilegios de administraodr o de usuario regular.

User					
Campo	Descripción	Tipo	Valores Permitidos	Restricciones	Ejemplo
name	Nombre del usuario	String	-	Requerido, No vacío	Juan
lastname	Apellido del usuario	String	-	Requerido, No vacío	Pérez
email	Correo electrónico único del usuario	String	-	Requerido, Formato Email	juan.perez@email.com
password	Contraseña del usuario	String	-	Requerido, Mínimo 6 caracteres	mypassword123
role	Rol del usuario en el sistema	String	user, admin, superuser	Opcional, Si se proporciona debe ser uno de los valores enumerados	user

Tabla 4.22: Diccionario de datos para la entidad de usuario, describiendo las columnas y su propósito en el modelo

A continuación, presentamos la estructura de la entidad *Simulation* (ver Tabla 4.23, la cual representa una simulación dentro del sistema. Cada campo en esta entidad es esencial para el correcto registro, seguimiento y análisis de las simulaciones ejecutadas. A continuación se detallan los atributos de la entidad y sus respectivas características:

- _id: Este campo representa el identificador único de la simulación. Al momento de crear un nuevo modelo de simulación y almacenarla en la base de datos de MongoDB Mongoose genera automáticamente este identificador, esto garantiza que cada simulación posea un identificador exclusivo en la base de datos. La generación automática del identificador asegura que no existan duplicados, lo cual es fundamental para mantener la integridad de los datos.

- name: El nombre de la simulación es un campo obligatorio, cuyo valor debe ser una cadena de caracteres con una longitud mínima de tres caracteres. Este atributo es necesario para identificar de manera única el modelo de simulación dentro del sistema y proporcionar un nombre descriptivo a cada nodo de la simulación almacenado en la base de datos de MongoDB.

- description: La descripción de la simulación es un campo obligatorio que permite proporcionar información detallada sobre el propósito y el alcance de la simulación. Este atributo es importante para asegurar que el usuario pueda identificar y entender el contexto de cada modelo de simulación ejecutada.

- nodes: Este campo contiene una lista de nodos que forman parte del modelo de simulación. Los nodos son elementos fundamentales en el modelo de simulación, y cada uno de ellos debe cumplir con el formato definido por el DTO nodeDto. El campo es obligatorio y debe incluir un arreglo no vacío de nodos, lo que asegura que cada modelo de simulación esté correctamente estructurado en términos de sus componentes básicos.

- edges: Al igual que nodos, los 'bordes' (edges) son esenciales para conectar los nodos en la simulación. Este campo también es obligatorio y debe contener un arreglo no vacío de bordes, cada uno de los cuales debe ajustarse al formato del DTO EdgeDto.

Los bordes permiten definir las relaciones y flujo de información entre los nodos en la simulación.

- status: El estado del modelo de simulación es un campo que indica en que etapa se encuentra la simulación. Los posibles valores de este campo son 'created', 'in progress', 'completed', 'failed', 'enqueued', 'dequeued', 'processing' y 'updated'. El valor por defecto al momento de crear un nuevo modelo de simulación es asignado como 'created' y este campo es obligatorio para reflejar de manera precisa el ciclo de vida de la simulación.
- user: Este campo contiene el identificador único del usuario que ha creado el modelo de simulación. El valor es obligatorio y debe corresponder al identificador único del usuario registrado en el sistema. Este campo permite asociar cada modelo de simulación a un usuario específico, facilitando del seguimiento y la gestión de los modelos de simulación por parte de los usuarios.
- results: El campo de resultados es opcional y se utiliza para almacenar los resultados generados por la simulación. Por defecto este campo es un arreglo vacío. Este atributo se actualizará una vez que la simulación haya finalizado, permitiendo almacenar los datos generados a lo largo del proceso de simulación.

Simulation					
Campo	Descripción	Tipo	Valores Permitidos	Restricciones	Ejemplo
_id	Identificador único de la simulación.	String	-	Creado automáticamente por Mongoose Mongoose no acepta claves duplicadas	6755edfce8dbd4f02947c81e
name	Nombre de la simulación.	String	-	Entrada Obligatoria Longitud mínima 3 caracteres	Simulación de tráfico
description	Descripción de la simulación.	String	-	Entrada Obligatoria	Simulación para analizar el tráfico en una red de 10 nodos
nodes	Conjunto de nodos que forman parte de la simulación.	Lista de Node	-	Entrada Obligatoria Arreglo no vacío Cada nodo debe cumplir con el DTO NodeDto	Ver descripción de la entidad Node
edges	Conjunto de bordes que conectan los nodos en la simulación.	Lista de Edge	-	Entrada Obligatoria Arreglo no vacío Cada borde debe cumplir con el DTO EdgeDto	Ver descripción de la entidad Edge
status	Estado de la simulación.	Enumeración	created, in progress, completed, failed, enqueued, dequeued, processing, updated	Entrada Obligatoria Por defecto 'created'	created
user	Identificador único del usuario	String (ID)	-	Entrada Obligatoria Debe ser un identificador único del usuario asociado	6756e87d82e3be4c8a12ebe7
results	Conjunto de resultados de una simulación	Lista de Objetos (Array)	-	Opcional por defecto: Arreglo vacío []	

Tabla 4.23: Diccionario de datos para la entidad de simulación, describiendo las columnas y su propósito en el modelo

A continuación, se describe la estructura de la entidad *Node* (ver tabla 4.24 encargada de representar un nodo dentro del sistema de simulación. Un nodo es un elemento fundamental que interactúa con otros nodos a través de bordes (edges) para formar una red

o un sistema. Cada campo en esta entidad proporciona información crítica sobre el nodo, su tipo, posición y otras propiedades asociadas:

- id: Este campo representa el identificador único del nodo, el cual se utiliza para distinguirlo de otros nodos en el sistema. El identificador es obligatorio y debe ser un valor único para cada nodo en la simulación. Este campo es crucial para asegurar la trazabilidad e integridad de los nodos en el sistema.

- type: El tipo de nodo especifica la categoría o clasificación del nodo dentro del sistema. Este campo también es obligatorio y permite diferenciar entre distintos tipos de nodos que hayan sido creado utilizando la librería de ReactFlow en el servicio del FrontEnd.

- data: El campo *data* contiene los detalles específicos y críticos sobre el nodo. Este campo es obligatorio y está diseñado para almacenar un objeto con información esencial relevante para el nodo. La estructura y valores específicos de este objeto están definidos por la entidad *data*, la cual describe en detalle las características personalizadas del nodo.

- position: Este campo define la posición del nodo en un espacio 2D, permitiendo representar gráficamente la distribución de los nodos en el modelo de la simulación. La posición es una propiedad obligatoria e importante para la visualización e interacción de los nodos dentro de la interfaz de usuario. El valor de este campo es un objeto que sigue la estructura de la entidad *position*.

- measured: Este campo especifica las dimensiones del nodo en el espacio 2D, como su tamaño o área de influencia dentro de la simulación. De carácter obligatorio y fundamental para representar visualmente la escala y la relación espacial de los nodos en el modelo de la simulación. La información de este campo está contenida en un objeto que sigue el formato de la entidad *Measured*

- selected: El campo selected es opcional y se utiliza para indicar si el nodo ha sido seleccionado por el usuario en la interfaz de usuario. Este atributo, de tipo booleano, permite gestionar interacciones como la selección de nodos para su manipuliacón o edición durante la simulación, su valor por defecto es *false*m indicando que el nodo no esta seleccionado.

- dragging: Similar al campo *selected*, *dragging* es un campo opcional que indica si el nodo está siendo arrastrado en la interfaz de usuario. Este atributo booleando se utiliza para gestionar las interacciones de arraste de los nodos, permitiendo a los usuario mover los nodos dentro del espacio 2D.

Node					
Campo	Descripción	Tipo	Valores Permitidos	Restricciones	Ejemplo
id	Identificador único del nodo	String	-	Requerido	B-1
type	Tipo de nodo	String	-	Requerido	custom
data	Detalles específicos del nodo	Objeto Data	-	Requerido	Ver descripción de la entidad data
position	Posición del nodo en el espacio 2D	Objeto Position	-	Requerido	Ver descripción de la entidad position
measured	Dimensiones del nodo en el espacio 2D	Objeto Measured	-	Requerido	Ver descripción de la entidad measured
selected	Indica si el nodo está seleccionado	Boolean	true, false	Opcional	false
dragging	Indica si el nodo esta siendo arrastrado	Boolean	true, false	Opcional	false

Tabla 4.24: Diccionario de datos para la entidad de nodo, describiendo las columnas y su propósito en el modelo

La siguiente tabla (ver tabla 4.25) presenta una descripción detallada de los campos utilizados en la configuración de los nodos del sistema. Cada campo corresponde a un atributo específico que define las características y comportamiento de los nodos dentro de la simulación. Los campos incluyen información sobre el nombre del nodo, el nivel de replicación, el tipo de agrupamiento que utilizará, la dirección IP del procesador y las distribuciones asociadas al tiempo de servicio, tasa de llegada y número de tweets de salida. A continuación, se detallan los campos de la entidad *data*:

- name: El nombre del nodo, utilizado para identificarlo dentro del sistema de simulación. Este campo es obligatorio debe ser un valor de tipo cadena. Un ejemplo común de nombre podría ser 'Nodo Spout', como se puede apreciar en el ejemplo.
- type: Especifica el tipo de nodo, que puede ser un *Spout* o un *Bolt*. Los valores permitidos son 'S' para Spout y 'B' para Bolt. Este campo es obligatorio y debe ser uno de estos dos valores, ya que cada tipo de nodo cumple una función distinta dentro de la simulación. En el ejemplo, se muestra 'S' como valor de tipo.
- replicationLevel: Indica el nivel de replicación del nodo, un valor numérico mayor o igual a 1 que determina cuántas instancias del nodo deben ser replicadas en la simulación. Este campo es obligatorio y garantiza que el nodo tenga la cantidad adecuada de réplicas para su funcionamiento.
- groupType: Define el tipo de agrupación del nodo, permitiendo clasificarlo dentro de uno de los grupo predefinidos. Los valores posibles son:
  - 0: Representa un agrupamiento de tipo Shuffle que itera sobre un mapa para enviar una replica de cada Bolt de destino, con un esquema de agrupamiento 'Roun Robin' donde las tuplas son distribuidas equitativamente entre un conjunto de nodos destino.
  - 1: Representa un agrupamiento de tipo Hash, que determina el nodo de destino para las tuplas

    - 2 y 3: Representan el mismo tipo de agrupamiento que retorna un valor específico.

- processor: Este campo especifica el procesador asociado al nodo, representado por una dirección IP del rango 10.0.0.0 a 10.3.1.254. Es un campo obligatorio que indica la ubicación del procesador encargado de ejecutar las operaciones del nodo en la simulación.

- avgServiceTimeType: Indica el tipo de distribución utilizado para modelar el tiempo promedio de servicio en el nodo. Los valores permitidos incluyen distribuciones específicas para este campo: spline, fixed, chi2, maxwell, expon, invgauss, norm y lognorm. Este campo permite personalizar cómo se modela el tiempo de servicio de cada nodo.

- avgServiceTimeValue: Especifica el valor del tiempo promedio de servicio. Este valor debe ser proporcionado en un formato adecuado y está asociado con el tipo de distribución especificado en el campo anterior.

- arrivalRateType: Similar al campo *avgServiceTimeType*, este campo define el mismo tipo de distribuciones que *avgServiceTimeType* para la tasa de llegada de eventos al nodo. Se utiliza para modelar cómo llegan los eventos al nodo.

- arrivalRateValue: Representa el valor de la tasa de llegada, configurado según el tipo de distribución elegido en arrivalRateType. Permite especificar la frecuencia de los eventos que llegan al nodo.

- numberOutputTweetsType: Define el tipo de distribución para la cantidad de tweets de salida generados por el nodo. Las distribuciones permitidas incluyen: spline, fixed, bernoulli, geom y nbinom. Este campo permite personalizar la cantidad de eventos de salida generados por el nodo.

- numberOutputTweetsValue: Especifica el valor de la cantidad de tweets de salida generados por el nodo. Permitiendo definir cuántos eventos de salida se generan en cada iteración durante la simulación.

data					
Campo	Descripción	Tipo	Valores Permitidos	Restricciones	Ejemplo
name	Nombre del nodo.	String	-	Requerido	Nodo Spout
type	Tipo de Nodo (Spout o Bolt).	String	'S', 'B'	Requerido, debe ser uno de los valores: S o B	S
replicationLevel	Nivel de replicación asociado al nodo.	Number	-	Requerido, ≥ 1	1
groupType	Tipo de agrupación del nodo.	String	'0', '1', '2', '3'	Requerido, uno de los valores predefinidos	0
processor	Procesador asociado al nodo.	String	-	Requerido, debe estar en el rango de IP 10.0.0.0 a 10.3.1.254	10.0.0.1
avgServiceTimeType	Tipo de distribución para el tiempo promedio de servicio.	String	'spline', 'fixed', 'chi2', 'maxwell', 'expon', 'invgauss', 'norm', 'lognorm'	Opcional	fixed
avgServiceTimeValue	Valor del tiempo promedio de servicio.	String	-	Opcional, en formato adecuado	fixed(4.0)
arrivalRateType	Tipo de distribución para la tasa de llegada.	String	'spline', 'fixed', chi2', 'maxwell', 'expon', 'invgauss', 'norm', 'lognorm'	Opcional	fixed
arrivalRateValue	Valor de la tasa de llegada.	String	-	Opcional	fixed(5.0)
numberOutputTweetsType	Tipo de distribución para la cantidad de tweets de salida.	String	'spline', 'fixed', 'bernoulli', 'geom', 'nbinom'	Opcional	fixed(4.0)
numberOutputTweetsValue	Valor de cantidad de tweets de salida.	String	-	Opcional	fixed(10.0)

Tabla 4.25: Diccionario de datos para la información detallada de cada nodo, especificando las características y atributos asociados

La tabla siguiente describe la entidad *position* (ver tabla 4.26, que se refiere a las coordenadas de un punto en el espacio 2D. Cada campo representa una coordenada específica que se usa para ubicar un nodo dentro de un plano. A continuación, se detallan los campos de esta entidad:

- x: Representa la coordenada X de un nodo en el espacio 2D. Este campo es obligatorio y debe ser un valor numérico que define la posición horizontal del nodo en el plano. Valor proporcionado en el ejemplo es 526.6666666666667
- y: Especifica la coordenada Y de un nodo en el espacio 2D. Al igual que la coordenada X, este campo es obligatorio y debe ser un valor numérico que define la posición vertical del nodo en el plano. En el ejemplo el valor proporcionado es 93.33333333333336.

El principal motivo por el cual se almacenan estos valores en la base de datos de MongoDB radica en la necesidad de poder recrear el modelo de simulación en la interfaz de usuario. Al almacenar el modelo, se asegura que, una vez guardado en la base de datos, pueda ser recreado correctamente en la interfaz. Esto se logra utilizando los valores de cada nodo, tal como fueron definidos al momento de su creación, dentro de un espacio 2D, empleando ReactFlow. Este enfoque permite una representación visual precisa y fiel del modelo en cualquier momento posterior a la creación de un modelo de simulación.

Position					
Campo	Descripción	Tipo	Valores Permitidos	Restricciones	Ejemplo
x	Coordenada X en el espacio 2D	Number	-	Requerido	526.6666666666667
y	Coordenada Y en el espacio 2D	Number	-	Requerido	93.33333333333336

Tabla 4.26: Diccionario de datos para las posiciones, describiendo los atributos y coordenadas asociadas a cada ubicación

En el contexto de la simulación, además de las posiciones de los nodos en el espacio 2D, es fundamental registrar las métricas relacionadas con las dimensiones de cada nodo. Estos valores permiten comprender mejor las características físicas de los nodos dentro del modelo. A continuación, se presenta un diccionario de datos que describe las métricas medidas, específicamente el ancho y la altura de cada nodo, atributos que son registrados durante la simulación para asegurar que la representación visual sea precisa y coherente con las dimensiones reales de los nodos. A continuación, se detallan los campos de esta entidad:

- width: Representa el ancho de un nodo. Este campo es obligatorio y debe ser un valor numérico que define la extensión horizontal del nodo. El valor proporcionado en el ejemplo es 320.
- height: Especifica la altura de un nodo. Al igual que el ancho, este campo es obligatorio y debe ser un valor numérico que define la extensión vertical del nodo. En el ejemplo, el valor proporcionado es 227.

Measured					
Campo	Descripción	Tipo	Valores Permitidos	Restricciones	Ejemplo
width	Ancho del nodo	Number	-	Requerido	320
height	Altura del nodo	Number	-	Requerido	227

Tabla 4.27: Diccionario de datos para las métricas medidas, describiendo los atributos registrados durante la simulación

La tabla siguiente describe la entidad *Edge* 4.28, que se refiere a las conexiones entre nodos dentro del modelo de simulación. Cada campo representa un atributo específico que describe una relación entre dos nodos en el espacio 2D. A continuación, se detallan los campos de esta entidad:

- source: Especifica el nodo fuente de la conexión. Este campo es obligatorio y debe ser un valor de tipo cadena (string) que identifica el nodo desde el cual se origina la conexión. El valor proporcionado en el ejemplo es 'S-0'

- target: Representa el nodo destino de la conexión. Este campo también es obligatorio y debe ser un valor de tipo cadena (string), que identifica el nodo al que se dirige la conexión. En el ejemplo, el valor proporcionado es 'B-1'
- type: Indica el tipo de conexión entre los nodos. Este campo es obligatorio y debe ser un valor de tipo cadena (string) que especifica el tipo de enlace. El valor de ejemplo es 'animatedSvgEdge'
- data: Contiene los detalles especificos de la conexión. Este campo es obligatorio y está representado por un objeto de tipo EdgeData. Podemos encontrar el detalle de esta entidad en el diccionario de datos de EdgeData.
- id: Es el identificador único de la conexión. Este campo es obligatorio y debe ser un valor de tipo cadena(string) que permite identificar de manera única cada conexión entre nodos. El ejemplo proporcionado es 'xy-edge__S-0-B1'

Edge					
Campo	Descripción	Tipo	Valores Permitidos	Restricciones	Ejemplo
source	Nodo fuente de la conexión.	String	-	Requerido	S-0
target	Nodo destino de la conexión.	String	-	Requerido	B-1
type	Tipo de conexión.	String	-	Requerido	animatedSvgEdge
data	Detalles de la conexión.	Objeto EdgeData	-	Requerido	Ver descripción de la entidad EdgeData
id	Identificador único de la conexión.	String	-	Requerido	xy-edge__S-0-B-1

Tabla 4.28: Diccionario de datos para las conexiones entre nodos (edges), detallando los atributos asociados a cada enlace

La siguiente tabla describe *EdgeData* 4.29), la cual contiene información detallada sobre las conexiones entre los nodos. Cada campo representa un atributo específico de la conexión, como la duración, forma y ruta. Estos atributos son esenciales para modelar cómo se interconectan los nodos en el espacio de simulación. A continuación, se detallan los campos de esta entidad:

- duration: Representa la duración duración de la conexión. Este campo es obligatorio y debe ser un valor numérico que especifica el tiempo en el cual la conexión permanece activa. En el ejemplo, el valor proporcionado es 3.
- shape: Define la forma de la conexión. Este campo es de tipo cadena y también es obligatorio. El valor proporcionado en el ejemplo es "package", que representa una forma específica de la conexión.
- path: Especifica la ruta que sigue la conexión. Similar a los otros campos, este es obligatorio y se define como una cadena. El ejemplo proporcionado es "smoothstep", que indica una ruta suave para la conexión.

EdgeData					
Campo	Descripción	Tipo	Valores Permitidos	Restricciones	Ejemplo
duration	Duración de la conexión	Number	-	Requerido	3
shape	Forma de la conexión	String	-	Requerido	package
path	Ruta de la conexión	String	-	Requerido	smoothstep

Tabla 4.29: Diccionario de campos de la tabla EdgeData, describiendo la duración, forma y ruta de la conexión asociados a cada enlace

La siguiente tabla describe la entidad *SimulationStart* (ver tabla 4.30), un objeto de transferencia de datos utilizado para validar los parámetros necesarios para iniciar una simulación. Esta entidad asegura que los valores proporcionados sean correctos y cumplan con las restricciones establecidas antes de ejecutar el proceso de simulación en el sistema. A continuación, se detallan los campos de esta entidad:

- time: Tiempo en unidades de tiempo de una simulación. Este campo campo es opcional, en caso de no ser proporcionado durante el proceso de simulación de un modelo de simulación, el sistema automáticamente asigna un valor por defecto igual a : 2.688×10^{43}
- qty_tuples: Número de tuplas que se deben generar durante la simulación de un modelo de simulación. Este campo es obligatorio y debe ser un número mayor que 0.

SimulationStart					
Campo	Descripción	Tipo	Valores Permitidos	Restricciones	Ejemplo
time	Tiempo para la simulación	Number	-	Opcional Debe ser mayor que 0 si se proporciona	10
qty_tuples	Cantidad de tuplas a generar para la simulación	Number	-	Requerido, No vacío	100

Tabla 4.30: Diccionario de datos para la entidad de simulationStart, describiendo las columnas y su propósito en el modelo

4.4. Diseño de Pruebas

4.4.1. Pruebas de Integración

Escenario de Pruebas de Integración del BFF

ID de prueba	**Caso de prueba**	Objetivo	Entrada	Resultado esperado
BC-01	Autenticación de usuario - Credenciales inválidas	Asegurar que el endpoint POST /auth/signin retorne un error si las credenciales son inválidas.	Credenciales de usuario inválidas (email incorrecto o contraseña incorrecta).	Se lanza una excepción UnauthorizedException con código de estado 401 y mensaje "Credenciales incorrectas".
BC-02	Autenticación de usuario - Token expirada	Verificar que el token JWT caduca correctamente y genera un error cuando se intenta acceder con un token expirado.	Un token JWT expirado en la cabecera Authorization.	Se lanza una excepción UnauthorizedException con código de estado 401 y mensaje "Token expirado".
BC-03	Autenticación de usuario - Token válido	Asegurar que el endpoint POST /auth/signin permita la autenticación con un token JWT válido.	Token JWT válido en la cabecera Authorization.	El usuario es autenticado correctamente y se obtiene acceso a las rutas protegidas.
BC-04	Obtención de usuario desde token	Verificar que el endpoint GET /users/ me recupere los datos del usuario autenticado utilizando el token JWT.	Token JWT válido en la cabecera Authorization.	Se devuelve el objeto de usuario correspondiente al token JWT.
BC-05	Acceso a recursos protegidos por roles	Asegurar que un usuario con un rol específico pueda acceder a rutas protegidas.	Token JWT con rol adecuado en la cabecera Authorization.	El usuario con el rol adecuado puede acceder a la ruta protegida, mientras que otros usuarios reciben un error de acceso.
BC-06	Acceso a recursos con rol incorrecto	Verificar que un usuario sin el rol adecuado sea denegado al intentar acceder a rutas protegidas.	Token JWT con rol incorrecto en la cabecera Authorization.	Se lanza una excepción ForbiddenException con código de estado 403 y mensaje "Acceso denegado".
BC-07	No permitir acceso sin token	Verificar que las rutas protegidas rechacen peticiones sin un token JWT válido.	Petición a una ruta protegida sin token en la cabecera Authorization.	Se lanza una excepción UnauthorizedException con código de estado 401 y mensaje "Token no proporcionado".
BC-08	No permitir acceso con token mal formado	Asegurar que el endpoint rechace un token JWT mal formado.	Token JWT mal formado en la cabecera Authorization (por ejemplo, token con formato incorrecto).	Se lanza una excepción UnauthorizedException con código de estado 401 y mensaje "Token mal formado".

Tabla 4.31: Escenario de pruebas para el BFF

Escenario de Pruebas de Integración del User Service

ID de prueba	Caso de prueba	Objetivo	Entrada	Resultado esperado
UC-01	Crear un nuevo usuario	Verificar que el endpoint POST /users cree un nuevo usuario correctamente.	Datos válidos de usuario (nombre, correo electrónico, contraseña)	El usuario es creado correctamente, se devuelve el objeto del usuario con un _id único.
UC-02	No permitir crear un usuario con correo electrónico duplicado	Asegurar que el endpoint POST /users no permita crear un usuario con un correo electrónico ya existente.	Datos de usuario con un correo electrónico duplicado	Se lanza una excepción HttpException con código de estado 400 BAD_REQUEST indicando que el correo ya existe.
UC-03	Obtener todos los usuarios	Verificar que el endpoint GET /users devuelva correctamente todos los usuarios.	Ninguna entrada	Se devuelve una lista de todos los usuarios almacenados en la base de datos.
UC-04	Obtener un usuario por correo electrónico	Verificar que el endpoint GET /users/by-email/:email recupere un usuario por su correo electrónico.	Correo electrónico válido	Se devuelve el usuario correspondiente al correo electrónico solicitado.
UC-05	Validar formato de correo electrónico al buscar usuario	Asegurar que el endpoint GET /users/by-email/:email valide correctamente el formato del correo electrónico.	Correo electrónico con formato inválido	Se lanza una excepción BadRequestException con el mensaje: "El formato del correo no es valido".
UC-06	Obtener un usuario por ID	Verificar que el endpoint GET /users/:id recupere un usuario por su ID.	ID de usuario válido	Se devuelve el usuario correspondiente al ID solicitado.
UC-07	Actualizar datos de un usuario	Verificar que el endpoint PATCH /users/:id actualice correctamente los datos de un usuario.	ID de usuario válido, datos de actualización (nombre, dirección, etc.)	El usuario se actualiza correctamente y se devuelve el usuario con los datos modificados.
UC-08	Actualizar la contraseña de un usuario	Verificar que el endpoint PUT /users/:id/password actualice correctamente la contraseña de un usuario.	ID de usuario válido, nueva contraseña	La contraseña del usuario se actualiza correctamente y se devuelve una confirmación de la actualización.
UC-09	Eliminar un usuario	Verificar que el endpoint DELETE /users/:id elimine correctamente un usuario de la base de datos.	ID de usuario válido	El usuario se elimina correctamente de la base de datos y se devuelve una confirmación de la eliminación.
UC-10	No permitir eliminar un usuario con ID inexistente	Asegurar que el endpoint DELETE /users/:id no permita eliminar un usuario que no exista.	ID de usuario inexistente	Se lanza una excepción HttpException con código de estado 404 NOT_FOUND, indicando que el usuario no existe.

Tabla 4.32: Pruebas de Integración para UsersService

Escenario de Pruebas de Integración del Simulation Service

ID de prueba	Caso de prueba	Objetivo	Entrada	Resultado esperado
SC-01	Crear una nueva simulación	Verificar que el endpoint POST /simulations cree una nueva simulación correctamente.	Datos válidos de simulación (parámetros de simulación, id de usuario)	La simulación es creada correctamente y se devuelve el objeto de simulación con un _id único.
SC-02	Obtener todas las simulaciones de un usuario	Verificar que el endpoint GET /simulations recupere correctamente todas las simulaciones asociadas a un usuario.	ID de usuario válido en el query (user_id)	Se devuelve una lista de todas las simulaciones asociadas al usuario con el user_id proporcionado.
SC-03	Obtener todas las simulaciones	Verificar que el endpoint GET /simulations/all recupere correctamente todas las simulaciones.	Ninguna entrada	Se devuelve una lista de todas las simulaciones almacenadas en la base de datos.
SC-04	Obtener una simulación por ID	Verificar que el endpoint GET /simulations/:id recupere correctamente una simulación por su ID.	ID de simulación válido	Se devuelve la simulación correspondiente al id solicitado.
SC-05	Actualizar una simulación	Verificar que el endpoint PUT /simulations/:id actualice correctamente los datos de una simulación.	ID de simulación válido, datos de actualización de simulación (parámetros de simulación)	La simulación se actualiza correctamente y se devuelve el objeto de simulación actualizado.
SC-06	Eliminar una simulación	Verificar que el endpoint DELETE /simulations/:id e limine correctamente una simulación.	ID de simulación válido	La simulación se elimina correctamente de la base de datos y se devuelve una confirmación de la eliminación.
SC-07	Actualizar el estado de una simulación	Verificar que el endpoint PATCH /simulations/:id/status actualice correctamente el estado de una simulación.	ID de simulación válido, nuevo estado	El estado de la simulación se actualiza correctamente y se devuelve una confirmación del cambio.
SC-08	Actualizar los resultados de una simulación	Verificar que el endpoint PATCH /simulations/:id/results actualice correctamente los resultados de una simulación.	ID de simulación válido, datos de resultados de simulación	Los resultados de la simulación se actualizan correctamente y se devuelve una confirmación de la actualización.

Tabla 4.33: Pruebas de Integración para SimulationService

Escenario de Pruebas de Integración del Producer Service

ID de prueba	Caso de prueba	Objetivo	Entrada	Resultado esperado
PC-01	Iniciar una simulación con datos válidos	Verificar que el endpoint POST /producer/:id/start inicie correctamente una simulación con datos válidos.	ID de simulación válido, objeto simulationData válido (datos de simulación), objeto startSimulationDto válido (datos para iniciar la simulación).	La simulación es procesada correctamente, se encola en RabbitMQ, y el servicio devuelve una confirmación del inicio de la simulación.
PC-02	Iniciar simulación con datos inválidos	Verificar que el endpoint POST /producer/:id/start maneje correctamente el caso en que los datos son inválidos.	ID de simulación válido, pero con datos de simulationData o startSimulationDto inválidos (por ejemplo, campos faltantes o con valores incorrectos).	Se lanza una excepción BadRequestException con el mensaje correspondiente sobre los datos inválidos.
PC-03	Encolamiento de simulación en RabbitMQ	Verificar que cuando se inicia una simulación con datos válidos, los datos sean correctamente encolados en RabbitMQ.	ID de simulación válido, datos de simulationData y startSimulationDto válidos.	Los datos de la simulación deben ser enviados a la cola de RabbitMQ correctamente, asegurándose de que el mensaje esté en la cola.
PC-04	Actualizar el estado de la simulación a través del HTTP	Verificar que el estado de la simulación se actualice correctamente al cambiar su estado a través de una solicitud HTTP.	ID de simulación válido, nuevo estado de la simulación en el cuerpo de la petición (por ejemplo, started, completed).	El estado de la simulación debe actualizarse correctamente y el servicio debe devolver un mensaje de éxito con un código 200 OK.
PC-05	Actualizar el estado con un estado inválido	Verificar que si se envía un estado inválido, se maneje correctamente el error y se devuelva una respuesta adecuada.	ID de simulación válido, estado inválido en el cuerpo de la solicitud (por ejemplo, estado no reconocido).	Se lanza una excepción BadRequestException indicando que el estado proporcionado no es válido.

Tabla 4.34: Pruebas de Integración para ProducerService

Escenario de Pruebas de Integración del Consumer Service

ID de prueba	Caso de prueba	Objetivo	Entrada	Resultado esperado
CS-01	Consumir un mensaje válido de RabbitMQ	Verificar que el servicio consuma correctamente un mensaje de RabbitMQ y procese los datos asociados a la simulación.	Mensaje encolado en RabbitMQ con datos válidos de simulación (por ejemplo, detalles de la simulación y parámetros).	El servicio consume el mensaje correctamente, procesa la simulación, y la simulación se marca como en progreso.
CS-02	Encolar un mensaje con datos inválidos	Verificar que el servicio maneje correctamente los mensajes inválidos de RabbitMQ y los marque como fallidos.	Mensaje encolado en RabbitMQ con datos inválidos (por ejemplo, campos faltantes o mal formateados).	El servicio captura el error, marca la simulación como fallida en la base de datos, y no intenta procesarla.
CS-03	Actualizar el estado de una simulación a través de HTTP	Verificar que el estado de una simulación se actualice correctamente después de consumir el mensaje.	Mensaje encolado en RabbitMQ, procesamiento exitoso, y la simulación debe tener un nuevo estado (por ejemplo, started).	El estado de la simulación se actualiza correctamente en el sistema a través de una solicitud HTTP con el nuevo estado.
CS-04	Actualizar el estado de simulación con un estado no válido	Verificar que el servicio maneje correctamente la actualización de estado con un valor inválido.	Mensaje encolado en RabbitMQ, procesamiento de la simulación, pero se intenta actualizar a un estado inválido.	Se lanza una excepción BadRequestException con un mensaje que indique que el estado no es válido.
CS-05	Consumir múltiples mensajes y procesarlos correctamente	Verificar que el servicio pueda consumir y procesar múltiples mensajes de RabbitMQ de manera concurrente o en lote.	Múltiples mensajes válidos encolados en RabbitMQ.	El servicio debe procesar cada mensaje de forma independiente, actualizando cada simulación correctamente.
CS-06	Simulación de errores durante el procesamiento y manejo de excepciones	Verificar que el servicio maneje correctamente los errores durante el procesamiento (por ejemplo, simulaciones fallidas).	Mensaje de simulación encolado con datos válidos, pero el procesamiento lanza una excepción (por ejemplo, error en base de datos).	El sistema debe capturar la excepción, registrar el error, y marcar la simulación como fallida sin interrumpir el flujo.

Tabla 4.35: Pruebas de Integración para ConsumerService

4.4.2. Pruebas de Sistema

ID de Prueba	Caso de Prueba	Objetivo	Entradas	Pasos	Resultado Esperado
PS-01	Simulación completa desde la creación hasta la actualización de resultados y estado	Validar el flujo completo desde la creación del usuario hasta la simulación completa, su procesamiento y actualización de resultados.	Crear un usuario con POST /users.	Crear un usuario: Llamada al endpoint POST /users.	El sistema debe permitir la creación de un usuario.
			Crear una simulación con POST /simulations.	Crear la simulación: Llamada al endpoint POST /simulations	La simulación debe ser creada y asociada al usuario.
			El ProducerService coloca el mensaje en RabbitMQ.	El ProducerService encola la simulación.	El ProducerService debe encolar la simulación en RabbitMQ.
			El ConsumerService consume el mensaje, procesa la simulación y actualiza el estado.	El ConsumerService consume el mensaje de RabbitMQ y actualiza el estado a ”in progress”.	El ConsumerService debe consumir el mensaje y cambiar el estado a ”in progress”.
			El ConsumerService actualiza el estado a ”completa” y almacena los resultados.	El ConsumerService actualiza los resultados de la simulación.	Los resultados deben ser almacenados correctamente en la simulación.
				El estado se actualiza a ”completa”.	El estado debe ser actualizado a ”completa”.
				Consultar la simulación con GET /simulations/:id para verificar el estado y resultados.	La consulta debe devolver la simulación con el estado correcto y los resultados.

Tabla 4.36: Pruebas de sistema

Capítulo 5

Implementación

En este capítulo se presenta de manera detallada el entorno tecnológico y los recursos utilizados en el desarrollo del proyecto, incluyendo las herramientas, software, lenguajes de programación y dispositivos empleados. Asimismo, se describe el proceso de implementación de los componentes finales del sistema, destacando la estructura del código desarrollado y las configuraciones aplicadas. Finalmente, se abordan las instancias y escenarios de prueba utilizados para validar el correcto funcionamiento de la solución propuesta.

5.1. Hardware utilizado

En esta sección se describe detalladamente el hardware empleado durante las etapas de desarrollo, pruebas y despliegue del proyecto. El objetivo principal fue garantizar que los componentes físicos y dispositivos seleccionados cumplieran con las exigencias de rendimiento, capacidad y escalabilidad requeridas para la implementación exitosa de la solución propuesta.

Durante el desarrollo, se utilizó una arquitectura híbrida que combinó recursos de la estación de trabajo local con tecnología de contenedorización mediante Docker [103]. Esta integración permitió aprovechar al máximo los recursos físicos disponibles para ejecutar entornos aislados, virtualizados y de alta eficiencia. Docker facilita la gestión y el despliegue de contenedores al compartir el kernel del sistema operativo del host, lo que reduce el consumo de recursos en comparación con máquinas virtuales convencionales [104]. Además, el almacenamiento de imágenes, datos persistentes y configuraciones en directorios locales optimiza la portabilidad y la administración de las aplicaciones en diversos entornos [105].

La configuración de hardware implementada fue seleccionada con el objetivo de ofrecer un rendimiento óptimo, garantizando la capacidad de ejecutar múltiples contenedores simultáneamente y de soportar las operaciones de desarrollo y pruebas sin interrupciones. A continuación, se detallan las especificaciones técnicas de la estación de trabajo local

utilizada:

- Procesador: Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz, 6 núcleos y 12 hilos.
- Memoria RAM: 8 GB DDR4 a 2933 MHz.
- Almacenamiento: 500 GB SSD (477 GB disponibles).
- Sistema Operativo: Windows 11 Home (versión 22H2) de 64 bits.

Esta configuración permitió garantizar la ejecución eficiente de los contenedores Docker, optimizando el desarrollo local y facilitando la integración y el despliegue de los servicios en las diferentes etapas del ciclo de vida del proyecto. Todos los componentes se ejecutan en la capa de Docker, donde el acceso al hardware es gestionado de manera eficiente por el gestor de recursos de Docker, lo que permite un uso equilibrado de la CPU, la memoria y el almacenamiento, sin comprometer el rendimiento general del sistema.

5.2. Software utilizado

En esta sección, se presenta una descripción detallada del software utilizado en el desarrollo del proyecto. Se abordan las herramientas y tecnologías fundamentales que respaldan cada etapa del proceso, desde el diseño y la implementación hasta la gestión de la infraestructura y el despliegue. La selección de cada herramienta se llevó a cabo en función de criterios específicos, tales como rendimiento, escalabilidad, facilidad de integración y su capacidad para satisfacer las necesidades del proyecto. A continuación, se describen las principales características y funcionalidades de cada software, junto con la justificación de su uso, destacando su idoneidad para garantizar un desarrollo eficiente y un resultado final de alta calidad.

5.2.1. Visual Studio Code

Visual Studio Code (VS Code) [106] es un entorno de desarrollo integrado (IDE) ligero, versátil y ampliamente adoptado por la comunidad de desarrollo de software. Se destaca por su rendimiento eficiente y su capacidad multiplataforma, lo que lo convierte en una opción ideal tanto para el desarrollo local como en entornos virtualizados o contenerizados utilizando Docker.

VS Code ofrece una amplia gama de características, que incluyen resaltado de sintaxis para múltiples lenguajes de programación, autocompletado inteligente mediante IntelliSense, depuración de código en tiempo real y terminal integrada. Además, facilita la integración con sistemas de control de versiones como Git, permitiendo un flujo de trabajo ágil y colaborativo [107].

Una de sus principales ventajas es su arquitectura extensible, que permite personalizar y ampliar sus capacidades mediante extensiones. Esto habilita la integración con herramientas y servicios en la nube, así como la incorporación de funcionalidades específicas para frameworks y tecnologías, adaptándose a las necesidades particulares del proyecto [106].

5.2.2. Docker y Docker Desktop

Docker es una plataforma que simplifica el desarrollo, envío y ejecución de aplicaciones mediante el uso de contenedores. Estos contenedores son entornos de software autocontenidos que encapsulan todo lo necesario para ejecutar una aplicación, incluyendo bibliotecas, dependencias y código fuente. Gracias a esta encapsulación, se facilita la creación de entornos de desarrollo consistentes y portátiles, eliminando problemas relacionados con discrepancias entre sistemas operativos y configuraciones [108].

Por otro lado, Docker Desktop es una aplicación diseñada para ejecutar Docker en sistemas operativos de escritorio, como Windows y macOS. Proporciona una interfaz gráfica intuitiva que permite a los desarrolladores gestionar contenedores, imágenes y volúmenes de manera eficiente. Además, facilita la creación, ejecución y depuración de contenedores a través de herramientas integradas y comandos simplificados [101].

Una característica destacada de Docker es su capacidad para integrar contenedores tanto en entornos locales como en la nube, lo que ofrece alta flexibilidad y escalabilidad en el despliegue de aplicaciones. Además, Docker Desktop facilita la integración con tecnologías como Kubernetes para la orquestación de contenedores, lo cual resulta fundamental en proyectos con requisitos complejos de despliegue y gestión [109].

5.2.3. NestJS

Para el desarrollo del backend, se utilizó NestJS, un framework de Node.js basado en TypeScript que facilita la creación de aplicaciones escalables y mantenibles. NestJS está diseñado alrededor de una arquitectura modular que permite estructurar el código de manera eficiente, organizando el proyecto en módulos, controladores y servicios. Este enfoque modular mejora la mantenibilidad y escalabilidad de la aplicación, permitiendo añadir nuevas funcionalidades sin afectar el rendimiento general del sistema. En particular, los servicios del proyecto fueron implementados utilizando las funcionalidades proporcionadas por NestJS, como la inyección de dependencias, lo que permitió un desarrollo ágil y una integración efectiva con otros componentes del sistema, como RabbitMQ para la mensajería asíncrona y MongoDB Atlas para la gestión de la base de datos. La integración de NestJS con estas tecnologías facilitó la creación de una arquitectura limpia y eficiente, adecuada para el manejo de aplicaciones distribuidas y de alta disponibilidad.

5.2.4. RabbitMQ

RabbitMQ es un sistema de mensajería de código abierto, ampliamente utilizado para la implementación de colas de mensajes en arquitecturas de aplicaciones distribuidas. Facilita la comunicación asíncrona entre los diferentes componentes del sistema, lo que optimiza la integración y mejora la escalabilidad de las aplicaciones. La plataforma ofrece una amplia gama de características robustas, como enrutamiento flexible, confirmaciones de entrega, clustering y soporte para diversos protocolos de comunicación. Una de las principales ventajas de RabbitMQ es su capacidad para gestionar grandes volúmenes de mensajes de manera eficiente, lo que lo convierte en una opción ideal para aplicaciones que requieren alta disponibilidad y un rendimiento óptimo. Además, su arquitectura flexible y modular permite su integración con una amplia variedad de tecnologías y sistemas, lo que lo hace adecuado para numerosos casos de uso, que van desde la mensajería entre microservicios hasta la integración de sistemas empresariales complejos.

5.2.5. MongoDB

MongoDB Atlas es una solución de base de datos como servicio (*DBaaS, por sus siglas en inglés*) proporcionada por MongoDB, que permite implementar bases de datos NoSQL en la nube de manera eficiente y escalable. Esta plataforma gestionada ofrece numerosas ventajas, como la alta disponibilidad, el rendimiento optimizado y la seguridad avanzada, lo que la convierte en una opción ideal para aplicaciones que requieren un acceso rápido y fiable a grandes volúmenes de datos. La implementación en la nube de MongoDB Atlas facilita la gestión de la infraestructura, ya que proporciona características automáticas como el escalado horizontal, copias de seguridad automáticas y actualizaciones sin tiempo de inactividad, lo que minimiza la carga operativa para los desarrolladores y mejora la resiliencia de la base de datos.

En el contexto de este proyecto, MongoDB Atlas ha sido integrado tanto en el user service como en el simulation service. El user service gestiona la información relacionada con los usuarios, permitiendo almacenar y consultar de manera eficiente datos como perfiles, credenciales y preferencias. Gracias a MongoDB Atlas, el user service puede escalar según la demanda y garantizar un acceso rápido a los datos de los usuarios sin comprometer el rendimiento o la disponibilidad.

Por otro lado, el simulation service se encarga de manejar los datos relacionados con las simulaciones, como resultados, configuraciones y estados de simulación. MongoDB Atlas ofrece un almacenamiento flexible y de alto rendimiento, lo que es fundamental para gestionar grandes volúmenes de datos generados durante el proceso de simulación. La capacidad de MongoDB Atlas para manejar estructuras de datos no relacionales, como documentos JSON, se adapta perfectamente a los requisitos del simulation service, ya que permite almacenar y consultar eficientemente los resultados complejos y dinámicos de las simulaciones.

5.3. Lenguajes de programación

En esta sección se presentan los lenguajes de programación utilizados en el desarrollo del proyecto. Se detalla el rol de cada lenguaje en la implementación de los diferentes componentes del sistema, así como su relevancia para el cumplimiento de los objetivos planteados. Además, se proporciona una descripción de las características clave de cada lenguaje y se justifican las razones detrás de su elección para este trabajo. A continuación, se describen los lenguajes de programación utilizados:

5.3.1. Typescript

TypeScript es un lenguaje de programación de código abierto desarrollado por Microsoft que extiende JavaScript al agregar un sistema de tipos estáticos opcional. Este lenguaje combina la flexibilidad y dinamismo de JavaScript con las ventajas de un tipado fuerte, lo que permite detectar errores en tiempo de desarrollo y mejorar la mantenibilidad del código. Su integración directa con el ecosistema de JavaScript lo convierte en una herramienta ideal para el desarrollo de aplicaciones modernas y escalables [110].

En el proyecto, TypeScript fue utilizado para el desarrollo de los servicios backend implementados con NestJS, así como en el frontend desarrollado con Next.js. En el backend, su uso permitió la creación de controladores, servicios y módulos bien definidos, con interfaces que garantizan la consistencia en la manipulación de datos y la comunicación entre componentes. En el frontend, TypeScript facilitó el desarrollo de componentes interactivos y reutilizables, garantizando la coherencia en la estructura de datos utilizada en las interfaces de usuario.

La elección de TypeScript se fundamenta en su capacidad para mejorar la productividad del desarrollo, reducir errores comunes en tiempo de ejecución y facilitar el trabajo colaborativo gracias a su sistema de tipos. Además, su amplia compatibilidad con herramientas modernas como Visual Studio Code y su fuerte adopción por la comunidad lo convierten en una opción confiable para proyectos de software de alta calidad.desarrolladores.

5.3.2. Typescript con React (.tsx)

TSX es una extensión de TypeScript que permite la integración de sintaxis JSX (JavaScript XML) para el desarrollo de interfaces de usuario en aplicaciones basadas en bibliotecas como React. Esta combinación proporciona un entorno poderoso que combina el tipado estático de TypeScript con la flexibilidad y expresividad de JSX, lo que facilita la creación de componentes dinámicos y reutilizables para el frontend de aplicaciones web.

En este proyecto, TSX fue utilizado en el desarrollo del frontend implementado con Next.js. Este enfoque permitió definir componentes React con un tipado explícito, garantizando consistencia en las propiedades recibidas y reduciendo errores relacionados con

datos mal estructurados. Gracias al soporte de TypeScript, los desarrolladores pudieron aprovechar las ventajas del autocompletado, la detección temprana de errores y la documentación de código, lo que mejoró significativamente la productividad y la calidad del desarrollo.

La elección de TSX se debió a su capacidad para optimizar la experiencia del desarrollador al trabajar con React, así como su compatibilidad con las herramientas modernas de desarrollo frontend. Esto permitió construir una interfaz de usuario robusta y eficiente, alineada con los objetivos del proyecto y las mejores prácticas de la industria.

5.4. Estrategia de implementación

Las estrategias de implementación desempeñaron un papel fundamental en el desarrollo del proyecto, permitiendo abordar de manera eficiente los desafíos técnicos y organizativos asociados con la creación de un sistema complejo y distribuido. Estas estrategias abarcan desde la elección de la arquitectura y herramientas tecnológicas hasta las metodologías empleadas para la gestión del trabajo y el despliegue del sistema. Cada decisión fue cuidadosamente tomada con el objetivo de garantizar la escalabilidad, la robustez y la mantenibilidad de la solución, así como para optimizar el rendimiento y la experiencia del usuario. A continuación, se describen las principales estrategias implementadas, destacando su impacto en la construcción del proyecto y los beneficios obtenidos.

5.4.1. Arquitectura Basada en Microservicios

El proyecto se desarrolló utilizando una arquitectura basada en microservicios, dividiendo el sistema en servicios independientes como el user-service, simulation-service, producer-service y consumer-service, cada uno con responsabilidades claramente definidas. Este enfoque permitió una mejor escalabilidad, facilidad de mantenimiento y la posibilidad de desplegar servicios de forma independiente. Para garantizar una comunicación eficiente y desacoplada entre los microservicios, se empleó RabbitMQ como broker de mensajes. Además, se seleccionó NestJS como framework principal debido a su estructura modular y su compatibilidad con herramientas modernas , lo que facilitó la implementación de controladores, servicios y validaciones.

5.4.2. Uso de Contenedores con Docker

Para garantizar consistencia en los entornos de desarrollo, pruebas y producción, se optó por Docker como herramienta principal de contenerización. Cada servicio fue encapsulado en un contenedor Docker independiente, asegurando que las configuraciones fueran idénticas en todas las etapas del desarrollo. Se crearon Dockerfiles optimizados para cada

servicio, y mediante docker-compose, se levantaron todos los servicios de manera conjunta, simplificando el desarrollo local y permitiendo simular el entorno de producción.

5.4.3. Implementación de Comunicación Asíncrona

RabbitMQ fue empleado como broker de mensajes para habilitar la comunicación asíncrona entre los servicios del sistema. Esta herramienta permitió gestionar de manera eficiente las cargas de mensajes, asegurando su entrega confiable incluso ante fallos temporales. Se configuraron colas específicas para cada flujo de trabajo, incluyendo una Dead Letter Queue (DLQ) para mensajes no procesados, los cuales se almacenaron en la base de datos con el estado ”failed” para su análisis posterior.

5.4.4. Patrones Estructurales Utilizados

Durante el desarrollo, se implementaron patrones de diseño estructurales que ayudaron a resolver problemas comunes y garantizar la mantenibilidad del sistema:

- Singleton: Utilizado en servicios como la conexión a RabbitMQ y MongoDB para garantizar que exista una única instancia compartida de estos recursos en todo el sistema, reduciendo el consumo innecesario de memoria y recursos.
- Facade: Implementado en el Backend for Frontend(BFF) para unificar las solicitudes del frontend y abstraer la complejidad de la comunicación entre servicios, simplificando la integración.
- Mediator: Empleado en la comunicación entre servicios mediante RabbitMQ, actuando como intermediario entre productores y consumidores para desacoplar las interacciones directas y manejar la mensajería de forma asíncrona.
- Dependency Injection: Utilizado de forma nativa en NestJS para gestionar las dependencias entre controladores, servicios y repositorios, promoviendo un diseño modular y fácil de probar.
- Composite: Utilizado para tratar a un grupo de objetos o clases de forma similar a un único objeto, formando jerarquías de componentes. Útil para modelar nodos de simulación y sus interacciones como una estructura compuesta para procesarlos de manera uniforme.

5.5. Simulador SimStream

5.5.1. Simulador SimStream y su Relevancia en la Implementación

El simulador SimStream [69] es una herramienta fundamental diseñada para modelar y evaluar plataformas de procesamiento de streams, permitiendo realizar análisis de rendimiento en aplicaciones que procesan datos en tiempo real. En este proyecto, SimStream no solo fue utilizado como referencia, sino que se escaló y adaptó para simular modelos y topologías de plataformas de procesamiento de streams en un entorno web, integrando servicios asociados. Esta herramienta resulta crucial para la implementación al proporcionar la capacidad de identificar cuellos de botella, optimizar recursos y predecir métricas clave como latencia y rendimiento. Al ser escalado, SimStream permitió simular y evaluar el comportamiento de plataformas distribuidas en un entorno controlado, mejorando así la toma de decisiones y el diseño de la arquitectura del sistema.

Plataformas de Procesamiento de Streams y su Aplicación en SimStream

El procesamiento de streams es una metodología distribuida diseñada para gestionar grandes volúmenes de datos provenientes de diversas fuentes, como redes sociales, sistemas de publicidad y aplicaciones científicas. Estas plataformas requieren procesar datos en tiempo real con baja latencia, alta tolerancia a fallos y escalabilidad, para ofrecer análisis precisos en cuestión de segundos.

SimStream utiliza este enfoque para modelar el comportamiento de plataformas como Apache Storm, reconocida por su capacidad de procesamiento en tiempo real y baja latencia. Apache Storm emplea unidades de procesamiento denominadas Processing Elements (PEs), las cuales se comunican mediante canales de datos explícitos para procesar y transferir la información. En el caso de SimStream, este modelo es utilizado para simular y evaluar aplicaciones de procesamiento de datos en tiempo real, como el análisis de tweets en una plataforma distribuida.

Topología de SimStream

SimStream permite modelar una aplicación de análisis de streams de Twitter, diseñada para procesar grandes volúmenes de datos a través de una topología de ocho unidades de procesamiento interconectadas. Estas unidades trabajan de forma descendente (*bottom-down*), procesando las tuplas de datos entrantes de manera eficiente. Los componentes principales del simulador incluyen:

- **Apache Storm:** Motor de procesamiento de streams que ofrece procesamiento en tiempo real con latencia mínima.

- **Clúster multi-núcleo:** Hardware simulado donde se despliegan las topologías de Storm, permitiendo analizar el uso de recursos como CPU y memoria.
- **Red de comunicación:** Representación del costo computacional asociado a la transmisión de datos entre nodos, tanto locales como remotos.

El simulador permite configurar el flujo de tareas, replicar procesos intensivos computacionalmente y analizar métricas clave como la utilización de recursos y los tiempos de respuesta bajo diferentes tasas de llegada de datos.

Detalles del Simulador y Red de Comunicación

Respecto a la implementación, SimStream utiliza un modelo de simulación orientado a procesos, Los procesos en el simulador son representados por unidades denominadas *Bolts/Spouts*, que son responsables del procesamiento de tuplas de datos. Los recursos en el simulador incluyen artefactos tales como los datos de los mensajes entrantes, variables globales (como las solicitudes de entrada de cada proceso), CPU, y redes de comunicación.

La simulación emplea la librería *libCppSim*, propuesta por el autor **Marzolla, Moreno** [111]. En este marco, cada proceso es implementado como una *corrutina*, que puede ser bloqueada o desbloqueada durante la simulación a través de operaciones como *hold()*, *passivate()* y *activate()*. Estas corrutinas representan el comportamiento concurrente y la interacción entre los diferentes componentes de la plataforma simulada.

En el contexto de la simulación, una corrutina C_i puede ser pausada durante un tiempo δ_t, que representa la duración de una tarea. Al expirar este tiempo de simulación, la corrutina C_i se activa automáticamente si se ha ejecutado previamente la operación *hold()*. En caso contrario, la corrutina C_i puede ser activada por otra corrutina C_j usando la operación *activate()*. Este mecanismo de activación y pausado facilita la representación de la interacción entre los diferentes componentes del sistema de procesamiento de streams.

La topología de la simulación se ejecuta en un clúster de procesadores, donde los nodos ubicados en el mismo procesador físico presentan un costo de comunicación menor en comparación con los nodos distribuidos en diferentes procesadores. Cada procesador dispone de recursos como CPU, RAM y memoria caché. La política de reemplazo utilizada para ambas memorias es *Least Recently Used (LRU)* [112], y el tamaño de estas memorias puede configurarse a través de los parámetros del simulador.

El simulador permite modelar topologías de aplicaciones distribuidas, en las cuales los nodos pueden asignarse a un mismo procesador físico o distribuirse entre varios, lo que afecta el costo de la comunicación. Los nodos pueden estar distribuidos de manera transparente a la aplicación, utilizando una red de comunicación simulada basada en la arquitectura Fat-Tree. Esta red está formada por switches y nodos de procesamiento organizados en PODs, cada uno de los cuales incluye nodos de procesamiento conectados a switches de diferentes capas (Core, Aggregation y Edge). Esta configuración favorece el

paralelismo, minimiza la congestión y proporciona tolerancia a fallos al ofrecer múltiples rutas para la transmisión de mensajes.

Validación y Resultados

El simulador SimStream fue validado comparando las métricas obtenidas en las simulaciones con las reportadas por una aplicación real implementada en Apache Storm. La comparación mostró un margen de error mínimo, lo que valida la precisión del simulador para predecir el comportamiento de aplicaciones similares en plataformas de procesamiento de streams.

SimStream se utilizó como banco de pruebas en el proyecto para evaluar el impacto de la replicación de procesos y la variación en las cargas de trabajo. Estas pruebas permitieron analizar la escalabilidad del sistema y su capacidad de respuesta ante un aumento en la tasa de llegada de datos, un factor crítico en aplicaciones de procesamiento de streams en tiempo real.

Importancia en la Implementación del Proyecto

Dentro del marco de la implementación de este proyecto, SimStream ofrece las siguientes ventajas:

- **Optimización de recursos:** El simulador permite experimentar con diferentes configuraciones para optimizar la asignación de recursos físicos como CPU y memoria en un entorno controlado, lo que facilita la toma de decisiones durante la implementación.
- **Identificación de cuellos de botella:** La simulación permite identificar los puntos críticos en el flujo de datos, lo cual es esencial para mejorar el rendimiento del sistema en la etapa de desarrollo.
- **Predicción de métricas:** SimStream proporciona predicciones de métricas clave como latencia y throughput, lo que permite anticipar el comportamiento del sistema bajo diferentes escenarios y ajustarlo antes de la implementación en producción.

Además, el enfoque modular de SimStream lo hace fácilmente adaptable a otras plataformas de procesamiento de streams, como Spark, Heron o Flink. Esto amplía su aplicabilidad más allá del caso de estudio presentado en el proyecto. Su inclusión en este proyecto no solo valida la arquitectura propuesta, sino que también guía las decisiones de diseño relacionadas con la comunicación entre servicios, la gestión de datos en tiempo real y la escalabilidad del sistema.

Capítulo 6

Validación del tratamiento

En esta sección se exponen, en primer lugar, las pruebas de concepto tecnológicas, las cuales permiten validas las tecnologías implementadas y facilitan su desarrollo. Posteriormente, se presentan las pruebas específicas de la solución desarrollada.

6.1. Estrategia de prueba

La estrategia de pruebas propuesta tiene como objetivo asegurar la calidad y fiabilidad del sistema desarrollado, a través de un enfoque estructurado que abarca los siguientes aspectos clave:

1. **Validación del comportamiento interno de los componentes individuales y su integración**: Se realizará una serie de pruebas unitarias e integrales para verificar que los componentes individuales, como controladores, servicios y módulos, funcionen correctamente. Esto incluye:

   - Validación de los endpoints de los controladores, asegurando que las solicitudes HTTP sean procesadas correctamente.
   - Verificación de la correcta implementación de los servicios, como el manejo de datos, la comunicación entre servicios y la integración con bases de datos y colas de mensajes.
   - Pruebas de integración entre los servicios del backend, validando que las interacciones entre servicios como user-service, simulation-service, producer-service y consumer-service se realicen de manera adecuada.

   Las pruebas de aceptación cubren casos de prueba fundamentales como la creación y actualización de simulaciones, la gestión de usuarios, la ejecución de simulaciones a través de RabbitMQ, y la visualización de resultados en el frontend. Es esencial que

todos los servicios estén correctamente integrados y comuniquen entre sí de manera eficaz.

2. **Cobertura de los flujos funcionales críticos del sistema**: Se asegura que los flujos esenciales del sistema, como la creación de simulaciones, la ejecución de simulaciones, la actualización de estados a través del 'producer-service', y el procesamiento de mensajes desde RabbitMQ en el 'consumer-service', sean correctamente cubiertos mediante pruebas funcionales. Los flujos funcionales clave que serán verificados incluyen:

   - **Creación de simulaciones**: Se verificará que el proceso de creación de una simulación funcione correctamente, incluyendo la validación de datos.
   - **Actualización de simulaciones**: Se validará que la actualización de datos de simulación, como los resultados o el estado, se ejecute correctamente y que no haya sobreescritura errónea de información.
   - **Encolamiento y consumo de mensajes**: Se probará que el 'producer-service' encole correctamente los mensajes y que el 'consumer-service' los consuma y procese adecuadamente, actualizando los resultados y el estado de las simulaciones.
   - **Visualización de resultados en el frontend**: Se garantizará que los resultados de las simulaciones puedan ser recuperados y visualizados correctamente en el frontend, asegurando que las peticiones al backend se manejen adecuadamente.

3. **Seguridad y robustez en los mecanismos de autenticación, autorización y manejo de errores**: Se pondrá un enfoque especial en asegurar que los mecanismos de autenticación y autorización sean sólidos, validando que las rutas sensibles estén protegidas adecuadamente. Además, se verificará que los sistemas de manejo de errores y excepciones sean robustos y funcionen según lo esperado. En particular:

   - **Autenticación y autorización**: Se validarán los mecanismos de autenticación (como JWT) y autorización en las rutas sensibles, garantizando que los usuarios no autenticados o no autorizados no puedan acceder a los recursos protegidos.
   - **Cifrado de contraseñas**: Se probará que el cifrado de contraseñas con bcrypt sea efectivo y que las contraseñas no se almacenen ni transmitan en texto plano.
   - **Manejo de errores y excepciones**: Se verificará que los errores (por ejemplo, en la creación de simulaciones, la conexión a RabbitMQ, etc.) sean manejados correctamente mediante excepciones controladas y respuestas adecuadas, como mensajes de error descriptivos o códigos de estado HTTP correctos (e.g., '400 Bad Request', '404 Not Found', '500 Internal Server Error').

- **Resiliencia en fallos**: Se probará que el sistema se comporte adecuadamente en situaciones de fallos, como la caída de servicios, la imposibilidad de conectarse a RabbitMQ o problemas con la base de datos, asegurando que el sistema recupere su estado y gestione los intentos de reintentos de manera eficiente.

4. **Pruebas de carga y rendimiento**: Además de las pruebas funcionales y de integración, se realizarán pruebas de rendimiento para garantizar que el sistema pueda manejar cargas de trabajo esperadas sin afectar su rendimiento, como la simulación de múltiples solicitudes concurrentes y el procesamiento eficiente de mensajes en RabbitMQ.

Esta estrategia de pruebas garantiza que todos los componentes del sistema funcionen de manera independiente y en conjunto, cubriendo tanto los flujos de negocio como los mecanismos de seguridad, manejo de errores y rendimiento. Se utilizarán pruebas de aceptación para verificar que los usuarios puedan interactuar correctamente con el sistema, mientras que las pruebas técnicas y de integración asegurarán que todos los servicios interactúan y funcionan correctamente dentro de la arquitectura general del sistema.

6.2. Resultados

6.2.1. Pruebas de Integración

ID de prueba	Prueba Realizada	Resultado Esperado	Resultado Obtenido
BC-01	Intentar autenticación con credenciales inválidas	Se lanza una excepción UnauthorizedException con código de estado 401 y mensaje "Credenciales incorrectas".	Se lanza una excepción UnauthorizedException con mensaje "Credenciales incorrectas".
BC-02	Intentar acceder con token expirado	Se lanza una excepción UnauthorizedException con código de estado 401 y mensaje "Token expirado".	Se lanza una excepción UnauthorizedException con mensaje "Token expirado".
BC-03	Acceder con token JWT válido	El usuario es autenticado correctamente y se obtiene acceso a las rutas protegidas.	El usuario es autenticado correctamente y se tiene acceso a las rutas protegidas.
BC-04	Obtener datos del usuario autenticado (GET /users/me)	Se devuelve el objeto de usuario correspondiente al token JWT.	Se devuelve el objeto de usuario correspondiente al token JWT.
BC-05	Acceder a recursos protegidos con rol adecuado	El usuario con el rol adecuado puede acceder a la ruta protegida, mientras que otros usuarios reciben un error de acceso.	El usuario con rol adecuado accede correctamente a la ruta protegida.
BC-06	Intentar acceder a recursos con rol incorrecto	Se lanza una excepción ForbiddenException con código de estado 403 y mensaje "Acceso denegado".	Se lanza una excepción ForbiddenException con mensaje "Acceso denegado".
BC-07	Intentar acceder sin token JWT	Se lanza una excepción UnauthorizedException con código de estado 401 y mensaje "Token no proporcionado".	Se lanza una excepción UnauthorizedException con mensaje "Token no proporcionado".
BC-08	Intentar acceder con token JWT mal formado	Se lanza una excepción UnauthorizedException con código de estado 401 y mensaje "Token mal formado".	Se lanza una excepción UnauthorizedException con mensaje "Token mal formado".

Tabla 6.1: Resultados de pruebas de Integración realizadas sobre el BFF

ID de prueba	**Caso de prueba**	Objetivo	Resultado esperado
UC-01	Crear un nuevo usuario	Verificar que el endpoint POST /users cree un nuevo usuario correctamente.	El usuario es creado correctamente, se devuelve el objeto del usuario con un _id único.
UC-02	No permitir crear un usuario con correo electrónico duplicado	Asegurar que el endpoint POST /users no permita crear un usuario con un correo electrónico ya existente.	Se lanza una excepción HttpException con código de estado 400 BAD_REQUEST indicando que el correo ya existe.
UC-03	Obtener todos los usuarios	Verificar que el endpoint GET /users devuelva correctamente todos los usuarios.	Se devuelve una lista de todos los usuarios almacenados en la base de datos.
UC-04	Obtener un usuario por correo electrónico	Verificar que el endpoint GET /users/by-email/:email recupere un usuario por su correo electrónico.	Se devuelve el usuario correspondiente al correo electrónico solicitado.
UC-05	Validar formato de correo electrónico al buscar usuario	Asegurar que el endpoint GET /users/by-email/:email valide correctamente el formato del correo electrónico.	Se lanza una excepción BadRequestException con el mensaje: "El formato del correo no es valido".
UC-06	Obtener un usuario por ID	Verificar que el endpoint GET /users/:id recupere un usuario por su ID.	Se devuelve el usuario correspondiente al ID solicitado.
UC-07	Actualizar datos de un usuario	Verificar que el endpoint PATCH /users/:id actualice correctamente los datos de un usuario.	El usuario se actualiza correctamente y se devuelve el usuario con los datos modificados.
UC-08	Actualizar la contraseña de un usuario	Verificar que el endpoint PUT /users/:id/password actualice correctamente la contraseña de un usuario.	La contraseña del usuario se actualiza correctamente y se devuelve una confirmación de la actualización.
UC-09	Eliminar un usuario	Verificar que el endpoint DELETE /users/:id elimine correctamente un usuario de la base de datos.	El usuario se elimina correctamente de la base de datos y se devuelve una confirmación de la eliminación.
UC-10	No permitir eliminar un usuario con ID inexistente	Asegurar que el endpoint DELETE /users/:id no permita eliminar un usuario que no exista.	Se lanza una excepción HttpException con código de estado 404 NOT_FOUND, indicando que el usuario no existe.

Tabla 6.2: Resultados de pruebas de Integración realizadas sobre el User Service

ID de prueba	Prueba Realizada	Resultado Esperado	Resultado Obtenido
SC-01	Crear una nueva simulación	Verificar que el endpoint POST /simulations cree una nueva simulación correctamente.	Simulación creada exitosamente.
SC-02	Obtener todas las simulaciones de un usuario	Verificar que el endpoint GET /simulations recupere correctamente todas las simulaciones asociadas a un usuario.	Todas las simulaciones del usuario recuperadas correctamente.
SC-03	Obtener todas las simulaciones	Verificar que el endpoint GET /simulations/all recupere correctamente todas las simulaciones.	Todas las simulaciones recuperadas correctamente.
SC-04	Obtener una simulación por ID	Verificar que el endpoint GET /simulations/:id recupere correctamente una simulación por su ID.	Simulación recuperada correctamente por ID.
SC-05	Actualizar una simulación	Verificar que el endpoint PUT /simulations/:id actualice correctamente los datos de una simulación.	Simulación actualizada correctamente.
SC-06	Eliminar una simulación	Verificar que el endpoint DELETE /simulations/:id e limine correctamente una simulación.	Simulación eliminada correctamente.
SC-07	Actualizar el estado de una simulación	Verificar que el endpoint PATCH /simulations/:id/status actualice correctamente el estado de una simulación.	Estado de simulación actualizado correctamente.
SC-08	Actualizar los resultados de una simulación	Verificar que el endpoint PATCH /simulations/:id/results actualice correctamente los resultados de una simulación.	Resultados de la simulación actualizados correctamente.

Tabla 6.3: Resultados de pruebas de Integración realizadas sobre Simulation Service

ID de prueba	**Caso de prueba**	Objetivo	Resultado esperado
PC-01	Iniciar una simulación con datos válidos	Verificar que el endpoint POST /producer/:id/start inicie correctamente una simulación con datos válidos.	La simulación es procesada correctamente, se encola en RabbitMQ, y el servicio devuelve una confirmación del inicio.
PC-02	Iniciar simulación con datos inválidos	Verificar que el endpoint POST /producer/:id/start maneje correctamente el caso en que los datos son inválidos.	Se lanza una excepción BadRequestException con el mensaje correspondiente sobre los datos inválidos.
PC-03	Encolamiento de simulación en RabbitMQ	Verificar que cuando se inicia una simulación con datos válidos, los datos sean correctamente encolados en RabbitMQ.	Los datos de la simulación deben ser enviados a la cola de RabbitMQ correctamente, asegurándose de que el mensaje esté allí.
PC-04	Actualizar el estado de la simulación a través del HTTP	Verificar que el estado de la simulación se actualice correctamente al cambiar su estado a través de una solicitud HTTP.	El estado de la simulación debe actualizarse correctamente y el servicio debe devolver un mensaje de éxito con un código 200 OK.
PC-05	Actualizar el estado con un estado inválido	Verificar que si se envía un estado inválido, se maneje correctamente el error y se devuelva una respuesta adecuada.	Se lanza una excepción BadRequestException indicando que el estado proporcionado no es válido.
UC-06	Obtener un usuario por ID	Verificar que el endpoint GET /users/:id recupere un usuario por su ID.	Se devuelve el usuario correspondiente al ID solicitado.
UC-07	Actualizar datos de un usuario	Verificar que el endpoint PATCH /users/:id actualice correctamente los datos de un usuario.	El usuario se actualiza correctamente y se devuelve el usuario con los datos modificados.
UC-08	Actualizar la contraseña de un usuario	Verificar que el endpoint PUT /users/:id/password actualice correctamente la contraseña de un usuario.	La contraseña del usuario se actualiza correctamente y se devuelve una confirmación de la actualización.
UC-09	Eliminar un usuario	Verificar que el endpoint DELETE /users/:id elimine correctamente un usuario de la base de datos.	El usuario se elimina correctamente de la base de datos y se devuelve una confirmación de la eliminación.
UC-10	No permitir eliminar un usuario con ID inexistente	Asegurar que el endpoint DELETE /users/:id no permita eliminar un usuario que no exista.	Se lanza una excepción HttpException con código de estado 404 NOT_FOUND, indicando que el usuario no existe.

Tabla 6.4: Resultados de pruebas de Integración realizadas sobre el Producer Service

ID de prueba	**Caso de prueba**	Objetivo	Resultado esperado
CS-01	Consumir un mensaje válido de RabbitMQ	Verificar que el servicio consuma correctamente un mensaje de RabbitMQ y procese los datos asociados a la simulación.	El servicio consume el mensaje correctamente, procesa la simulación, y la simulación se marca como en progreso.
CS-02	Encolar un mensaje con datos inválidos	Verificar que el servicio maneje correctamente los mensajes inválidos de RabbitMQ y los marque como fallidos.	El servicio captura el error, marca la simulación como fallida en la base de datos, y no intenta procesarla.
CS-03	Actualizar el estado de una simulación a través de HTTP	Verificar que el estado de una simulación se actualice correctamente después de consumir el mensaje.	El estado de la simulación se actualiza correctamente en el sistema a través de una solicitud HTTP con el nuevo estado.
CS-04	Actualizar el estado de simulación con un estado no válido	Verificar que el servicio maneje correctamente la actualización de estado con un valor inválido.	Se lanza una excepción BadRequestException con un mensaje que indique que el estado no es válido.
CS-05	Consumir múltiples mensajes y procesarlos correctamente	Verificar que el servicio pueda consumir y procesar múltiples mensajes de RabbitMQ de manera concurrente o en lote.	El servicio debe procesar cada mensaje de forma independiente, actualizando cada simulación correctamente.
CS-06	Simulación de errores durante el procesamiento y manejo de excepciones	Verificar que el servicio maneje correctamente los errores durante el procesamiento (por ejemplo, simulaciones fallidas).	El sistema debe capturar la excepción, registrar el error, y marcar la simulación como fallida sin interrumpir el flujo.

Tabla 6.5: Resultados de pruebas de Integración realizadas sobre el Consumer Service

Capítulo 7

Implantación

7.1. Implantación

El capítulo de implantación tiene como objetivo detallar el proceso de implementación del sistema propuesto, describiendo las etapas necesarias para su despliegue en un entorno de producción. La implantación del sistema debe llevarse a cabo siguiendo un enfoque estructurado que permita la integración de todos los componentes de manera eficiente y efectiva. Este proceso incluye la configuración y puesta en marcha de los servicios, la implementación de las soluciones de almacenamiento de datos, la configuración de colas de mensajes como RabbitMQ para la comunicación entre los distintos módulos, y la integración de las diversas interfaces que componen la aplicación. Además, se debe contemplar un ciclo de pruebas exhaustivas para verificar la funcionalidad del sistema, así como garantizar su escalabilidad y robustez en el entorno de producción. A lo largo de este capítulo, se presentarán las distintas fases del proceso de implantación, con un enfoque en la optimización y automatización de las tareas necesarias para la puesta en marcha exitosa del sistema.

7.1.1. Requerimientos

Requerimientos de Mínimos

- **Docker Engine:** v27.3.1
- **Docker Compose:** v2.30.3
- **CPU:** Procesador de 2 núcleos como mínimo (recomendado 4 núcleos)
- **Memoria RAM:** 4 GB (recomendado 8 GB)
- **Espacio en disco:** 10 GB libres

- **Sistema operativo:** Windows 10 o superior (recomendado WSL 2 en caso de utilizar Windows)

Requerimientos de Recomendados

- **Docker Engine:** v27.3.1 o superior
- **Docker Compose:** v2.30.3 o superior
- **CPU:** Procesador de 4 núcleos o más
- **Memoria RAM:** 8 GB o más
- **Espacio en disco:** 20 GB libres o más
- **WSL(Window Subsystem Linux):** 2

7.1.2. Preparación de Ambiente

La preparación del ambiente es un paso fundamental para asegurar que todos los componentes del sistema funcionen correctamente. Puesto que el proyecto de trabajo de título se realizo utilizando Docker es necesario realizar la correcta instalación y configuración del entorno para su ejecución. A continuación, se detallan los pasos necesarios para configurar y preparar el entorno adecuado para ejecutar el sistema utilizando Docker.

Instalación de Docker y Docker Compose

Para poder ejecutar el sistema en contenedores, es necesario tener instalado Docker y Docker Compose. Asegúrese de seguir los siguientes pasos para su instalación:

- **Instalación de Docker:**
  - Descargue la última versión de Docker desde `https://www.docker.com/products/docker-desktop`.
  - Siga las instrucciones para su instalación según el sistema operativo (Windows, macOS o Linux).
  - Asegúrese de que Docker esté correctamente instalado ejecutando el siguiente comando en la terminal:

    ```
    docker --version
    ```

    Esto debería devolver la versión instalada de Docker.

- **Instalación de Docker Compose:**
  - Docker Compose ya viene incluido en las versiones más recientes de Docker Desktop, pero si necesita instalarlo manualmente, siga las instrucciones en `https://docs.docker.com/compose/install/`.
  - Verifique la instalación ejecutando:

```
docker compose --version
```

Configuración del Entorno de Desarrollo

Para facilitar la ejecución y gestión del sistema en contenedores Docker, es recomendable configurar un entorno adecuado. Esto incluye la configuración de volúmenes, redes y variables de entorno necesarias para que los contenedores interactúen entre sí de manera eficiente.

- **Configuración de Docker Compose:**
  - Clone el repositorio del proyecto en su máquina local: `git clone https://github.com/bastiantoledosalas/App-web-for-processing-streams-platforms.git`
  - Navegue a la carpeta del proyecto donde se encuentra el archivo `docker-compose.yml`.
  - Asegúrese de que todos los servicios necesarios estén definidos correctamente en el archivo `docker-compose.yml`, incluyendo los contenedores para el simulador, RabbitMQ, MongoDB, entre otros.
- **Variables de Entorno:**
  - Algunas configuraciones, como la base de datos y las claves de API, pueden necesitar ser definidas como variables de entorno en el archivo `.env`.
  - Cree o edite el archivo `.env` con los valores correctos para las variables que se requieren para su configuración local.

Verificación del Entorno

Una vez que haya completado la instalación y configuración del entorno, verifique que todo esté en orden ejecutando los siguientes comandos:

- Inicie los contenedores de Docker Compose:

```
docker compose up -d
```

Esto iniciará todos los servicios definidos en el archivo `docker-compose.yml` en segundo plano.

- Verifique que los contenedores estén ejecutándose correctamente:

```
docker ps
```

Este comando le mostrará todos los contenedores activos. Asegúrese de que los contenedores para el simulador, RabbitMQ y MongoDB estén en ejecución.

Con estos pasos, el entorno estará listo para la ejecución del sistema. Asegúrese de tener suficiente memoria y espacio en disco para evitar posibles errores durante el proceso de ejecución.

7.1.3. Documentación

Manual de Usuario

La documentación completa y el manual de usuario de este proyecto se encuentran disponibles en el repositorio de GitHub. Estos documentos proporcionan instrucciones detalladas sobre la instalación, el uso, las configuraciones necesarias y las mejores prácticas para aprovechar al máximo la aplicación. Además, incluyen ejemplos y casos de uso para facilitar la comprensión y la implementación. Puedes acceder a la documentación y al manual de usuario en el siguiente enlace de GitHub: `https://github.com/bastiantoledosalas/App-web-for-processing-streams-platforms.git`

7.1.4. Documentación de desarrollo

Para mas documentación disponible en el repositorio: `https://github.com/bastiantoledosalas/App-web-for-processing-streams-platforms.git`

Capítulo 8

Conclusiones

8.1. Conclusiones

En el contexto de la simulación de plataformas de procesamiento de streams, se ha diseñado e implementado un entorno web basado en microservicios y contenedores Docker. Este enfoque ha permitido crear un sistema modular, escalable y flexible, capaz de ejecutar simulaciones en un entorno controlado y eficiente. La integración de RabbitMQ ha sido clave para gestionar la comunicación asincrónica entre los distintos componentes del sistema, garantizando alta disponibilidad y fiabilidad.

A lo largo del trabajo, se ha logrado integrar un simulador en C++ dentro de un contenedor Docker, el cual se gestiona a través de una API RESTful creada con FastAPI. Esta solución no solo ha facilitado la ejecución eficiente de simulaciones, sino que también ha expuesto los servicios a través de una interfaz web, simplificando su integración y uso por parte de otros sistemas y usuarios.

A pesar de su funcionalidad, el sistema actual presenta varios desafíos y áreas de mejora. La gestión de recursos en entornos de alta carga y la optimización de la interfaz de usuario para hacerla más interactiva y amigable son aspectos clave a mejorar. Además, se identificaron limitaciones en la capacidad de gestionar simulaciones a gran escala, lo que podría solucionarse mediante la integración con plataformas de orquestación como Kubernetes, permitiendo escalar el sistema horizontalmente.

8.1.1. Mejoras Futuras

El sistema presenta diversas oportunidades para su mejora y evolución:

- **Optimización de rendimiento:** Se puede mejorar el uso de recursos, especialmente en la gestión de grandes volúmenes de datos, mediante técnicas como la paralelización de procesos o el uso de contenedores más ligeros.

- **Integración de análisis en tiempo real:** Actualmente, el sistema realiza simulaciones en batch. Se podría integrar la capacidad de análisis en tiempo real para la monitorización constante de las simulaciones y la entrega de resultados instantáneos.
- **Escalabilidad horizontal:** La integración de un sistema de orquestación de contenedores como Kubernetes permitiría escalar los servicios de simulación, distribuyendo la carga entre varios nodos y mejorando la gestión de múltiples simulaciones simultáneas.
- **Mejoras en la interfaz de usuario:** El entorno web podría beneficiarse de una interfaz más intuitiva y visual. La inclusión de dashboards interactivos para visualizar el estado de las simulaciones en tiempo real podría mejorar significativamente la experiencia del usuario.
- **Soporte para más plataformas de procesamiento de streams:** Ampliar el sistema para soportar plataformas como Apache Kafka o Apache Flink permitiría adaptarse a diferentes necesidades y casos de uso.

8.1.2. Direcciones Futuras

El entorno web tiene un gran potencial para expandirse en diversas direcciones. La inclusión de capacidades de inteligencia artificial y aprendizaje automático podría optimizar las simulaciones de manera automática, basándose en los resultados obtenidos en tiempo real y ofreciendo recomendaciones para mejorar el rendimiento del sistema. Además, la integración con sistemas de big data permitiría procesar simulaciones de plataformas a gran escala, manejando grandes volúmenes de datos de manera eficiente.

El sistema también podría evolucionar hacia una plataforma de simulación en la nube, utilizando servicios como AWS, Google Cloud o Azure, proporcionando escalabilidad infinita y alta disponibilidad.

En resumen, el proyecto ha establecido una base sólida, pero las oportunidades para mejorar y expandir el sistema son vastas. Con estas mejoras y nuevas direcciones, el entorno web podría convertirse en una herramienta mucho más potente y flexible, capaz de abordar una amplia variedad de necesidades en simulación y procesamiento de streams.

Bibliografía

[1] M. Khalid and M. M. Yousaf, "A comparative analysis of big data frameworks: An adoption perspective," *Applied Sciences*, vol. 11, no. 22, p. 11033, 2021.

[2] T. Rabl, J. Traub, A. Katsifodimos, and V. Markl, "Apache flink in current research," *it-Information Technology*, vol. 58, no. 4, pp. 157–165, 2016.

[3] M. A. Lopez, A. G. P. Lobato, and O. C. M. Duarte, "A performance comparison of open-source stream processing platforms," in *2016 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2016, pp. 1–6.

[4] PhoenixNAP, "Apache Storm," https://phoenixnap.com/kb/apache-storm, accedido el 12 de marzo de 2024.

[5] S. Salloum, R. Dautov, X. Chen, P. X. Peng, and J. Z. Huang, "Big data analytics on apache spark," *International Journal of Data Science and Analytics*, vol. 1, pp. 145–164, 2016.

[6] Apache Spark Documentation. Cluster Overview - Apache Spark Documentation. [Online]. Available: https://spark.apache.org/docs/latest/cluster-overview.html

[7] M. Zaharia, T. Das, H. Li, T. Hunter, S. Shenker, and I. Stoica, "Discretized streams: fault-tolerant streaming computation at scale," p. 423–438, 2013. [Online]. Available: https://doi.org/10.1145/2517349.2522737

[8] B. Babcock and S. Babu, "Models and issues in data stream systems," *Proceedings of the 21st ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pp. 1–16, 2002.

[9] T. Barnett, S. Jain, U. Andra, and T. Khurana, "Cisco visual networking index (vni) complete forecast update, 2017–2022," *Americas/EMEAR Cisco Knowledge Network (CKN) Presentation*, pp. 1–30, 2018.

[10] B. Dab, I. Fajjari, N. Aitsaadi, and G. Pujolle, "Vnr-ga: Elastic virtual network reconfiguration algorithm based on genetic metaheuristic," in *2013 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2013, pp. 2300–2306.

[11] Apache Software Foundation. (Fecha de acceso: Febrero 2024) Apache kafka. [Online]. Available: https://kafka.apache.org/

[12] N. Garg, *Apache kafka*. Packt Publishing Birmingham, UK, 2013.

[13] Apache Software Foundation, "Apache flink," *Sitio web de Apache Flink*, Fecha de acceso: Febrero 2024. [Online]. Available: https://flink.apache.org/

[14] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas, "Apache flink: Stream and batch processing in a single engine," *The Bulletin of the Technical Committee on Data Engineering*, vol. 38, no. 4, 2015.

[15] Apache Software Foundation, "Apache spark," *Sitio web de Apache Spark*, Fecha de acceso: Febrero 2024. [Online]. Available: https://spark.apache.org/

[16] S. Kane and K. Matthias, *Docker: Up & Running*. O'Reilly Media, 2023. [Online]. Available: https://books.google.cl/books?id=hWC5EAAAQBAJ

[17] J. Schwaber, *Scrum: The Art of Doing Twice the Work in Half the Time*. Currency, 2017.

[18] D. J. Abadi, Y. Ahmad, M. Balazinska, U. Cetintemel, M. Cherniack, J.-H. Hwang, W. Lindner, A. Maskey, A. Rasin, E. Ryvkina *et al.*, "The design of the borealis stream processing engine." in *Cidr*, vol. 5, no. 2005, 2005, pp. 277–289.

[19] D. J. Abadi, D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik, "Aurora: a new model and architecture for data stream management," *the VLDB Journal*, vol. 12, pp. 120–139, 2003.

[20] A. Arasu, B. Babcock, S. Babu, J. Cieslewicz, M. Datar, K. Ito, R. Motwani, U. Srivastava, and J. Widom, "Stream: The stanford data stream management system," *Data Stream Management: Processing High-Speed Data Streams*, pp. 317–336, 2016.

[21] R. S. Barga, J. Goldstein, M. Ali, and M. Hong, "Consistent streaming through time: A vision for event stream processing," *arXiv preprint cs/0612115*, 2006.

[22] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. R. Madden, F. Reiss, and M. A. Shah, "Telegraphcq: continuous dataflow processing," in *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, 2003, pp. 668–668.

[23] C. Cranor, T. Johnson, O. Spataschek, and V. Shkapenyuk, "Gigascope: A stream database for network applications," in *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, 2003, pp. 647–651.

[24] T. Akidau, R. Bradshaw, C. Chambers, S. Chernyak, R. J. Fernández-Moctezuma, R. Lax, S. McVeety, D. Mills, F. Perry, E. Schmidt *et al.*, "The dataflow model: a practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing," *Proceedings of the VLDB Endowment*, vol. 8, no. 12, pp. 1792–1803, 2015.

[25] M. Fragkoulis, P. Carbone, V. Kalavri, and A. Katsifodimos, "A survey on the evolution of stream processing systems," *The VLDB Journal*, vol. 33, no. 2, pp. 507–541, 2024. [Online]. Available: https://rdcu.be/dBEoB

[26] M. Rychly, P. Smrz *et al.*, "Scheduling decisions in stream processing on heterogeneous clusters," in *2014 Eighth International Conference on Complex, Intelligent and Software Intensive Systems*. IEEE, 2014, pp. 614–619.

[27] H. C. Andrade, B. Gedik, and D. S. Turaga, *Fundamentals of stream processing: application design, systems, and analytics*. Cambridge University Press, 2014.

[28] J. Kreps, *I Heart Logs: Event Data, Stream Processing, and Data Integration*. O'Reilly Media, 2014.

[29] O. Etzion and P. Niblett, *Event processing in action*. Manning Publications Co., 2010.

[30] J. Boubeta-Puig, G. Díaz, H. Macià, V. Valero, and G. Ortiz, "Medit4cep-cpn: An approach for complex event processing modeling by prioritized colored petri nets," *Information systems*, vol. 81, pp. 267–289, 2019.

[31] W. A. Higashino, M. A. Capretz, and L. F. Bittencourt, "Cepsim: Modelling and simulation of complex event processing systems in cloud environments," *Future Generation Computer Systems*, vol. 65, pp. 122–139, 2016.

[32] T.-D. Nguyen and E.-N. Huh, "Ecsim++: An inet-based simulation tool for modeling and control in edge cloud computing," in *2018 IEEE International Conference on Edge Computing (EDGE)*, 2018, pp. 80–86.

[33] G. Amarasinghe, M. D. de Assuncao, A. Harwood, and S. Karunasekera, "Ecsnet++: A simulator for distributed stream processing on edge and cloud environments," *Future Generation Computer Systems*, vol. 111, pp. 401–418, 2020.

[34] P. Muthukrishnan and K. Fadnis, "Discrete event simulation based performance evaluation of stream processing systems," in *2020 IEEE International Conference on Big Data (Big Data)*. IEEE, 2020, pp. 1089–1098.

[35] Apache Software Foundation, "Apache samza," *Sitio web de Apache Samza*, Fecha de acceso: Febrero 2024. [Online]. Available: https://samza.apache.org/

[36] ——, "Apache storm," *Sitio web de Apache Storm*, Fecha de acceso: Febrero 2024. [Online]. Available: https://storm.apache.org/

[37] S. A. Noghabi, K. Paramasivam, Y. Pan, N. Ramesh, J. Bringhurst, I. Gupta, and R. H. Campbell, "Samza: stateful scalable stream processing at linkedin," *Proceedings of the VLDB Endowment*, vol. 10, no. 12, pp. 1634–1645, 2017.

[38] A. W. Services, "Aws kinesis," https://aws.amazon.com/kinesis/, accedido el 12 de marzo de 2024.

[39] Microsoft, "Azure eventhub," https://azure.microsoft.com/en-us/services/event-hubs/, accedido el 12 de marzo de 2024.

[40] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth *et al.*, "Apache hadoop yarn: Yet another resource negotiator," in *Proceedings of the 4th annual Symposium on Cloud Computing*, 2013, pp. 1–16.

[41] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica, "Mesos: a platform for fine-grained resource sharing in the data center," in *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'11. USA: USENIX Association, 2011, p. 295–308.

[42] P. Carbone, G. Fóra, S. Ewen, S. Haridi, and K. Tzoumas, "Lightweight asynchronous snapshots for distributed dataflows," *arXiv preprint arXiv:1506.08603*, 2015.

[43] S. Ewen, K. Tzoumas, M. Kaufmann, and V. Markl, "Spinning fast iterative data flows," *arXiv preprint arXiv:1208.0088*, 2012.

[44] B. Saha, H. Shah, S. Seth, G. Vijayaraghavan, A. Murthy, and C. Curino, "Apache tez: A unifying framework for modeling and building data processing applications," in *Proceedings of the 2015 ACM SIGMOD international conference on Management of Data*, 2015, pp. 1357–1369.

[45] J. Kreps, N. Narkhede, J. Rao *et al.*, "Kafka: A distributed messaging system for log processing," in *Proceedings of the NetDB*, vol. 11, no. 2011. Athens, Greece, 2011, pp. 1–7.

[46] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *2010 IEEE 26th symposium on mass storage systems and technologies (MSST)*. Ieee, 2010, pp. 1–10.

[47] A. Toshniwal, S. Taneja, A. Shukla, K. Ramasamy, J. M. Patel, S. Kulkarni, J. Jackson, K. Gade, M. Fu, J. Donham *et al.*, "Storm@ twitter," in *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, 2014, pp. 147–156.

[48] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed, "{ZooKeeper}: Wait-free coordination for internet-scale systems," in *2010 USENIX Annual Technical Conference (USENIX ATC 10)*, 2010.

[49] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauly, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A {Fault-Tolerant} abstraction for {In-Memory} cluster computing," in *9th USENIX symposium on networked systems design and implementation (NSDI 12)*, 2012, pp. 15–28.

[50] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen *et al.*, "Mllib: Machine learning in apache spark," *The journal of machine learning research*, vol. 17, no. 1, pp. 1235–1241, 2016.

[51] J. E. Gonzalez, "From graphs to tables the design of scalable systems for graph analytics," in *Proceedings of the 23rd International Conference on World Wide Web*, ser. WWW '14 Companion. New York, NY, USA: Association for Computing Machinery, 2014, p. 1149–1150. [Online]. Available: https://doi.org/10.1145/2567948.2580059

[52] R. S. Xin, J. E. Gonzalez, M. J. Franklin, and I. Stoica, "Graphx: A resilient distributed graph system on spark," in *First international workshop on graph data management experiences and systems*, 2013, pp. 1–6.

[53] M. Armbrust, R. S. Xin, C. Lian, Y. Huai, D. Liu, J. K. Bradley, X. Meng, T. Kaftan, M. J. Franklin, A. Ghodsi, and M. Zaharia, "Spark sql: Relational data processing in spark," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 1383–1394. [Online]. Available: https://doi.org/10.1145/2723372.2742797

[54] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O'Malley, S. Radia, B. Reed, and E. Baldeschwieler, "Apache hadoop yarn: yet another resource negotiator," in *Proceedings of the 4th Annual Symposium on Cloud Computing*, ser. SOCC '13. New York, NY, USA: Association for Computing Machinery, 2013. [Online]. Available: https://doi.org/10.1145/2523616.2523633

[55] A. J. Park, C.-H. Li, R. Nair, N. Ohba, U. Shvadron, A. Zaks, and E. Schenfeld, "Flow: A stream processing system simulator," in *2010 IEEE Workshop on Principles of Advanced and Distributed Simulation*, 2010, pp. 1–9.

[56] M. Kaptein, "Rstorm: Developing and testing streaming algorithms in r," *The R Journal*, vol. 6, no. 1, pp. 123–132, 2014.

[57] Z. Fang and H. Yu, "Parallel simulation of large-scale distributed stream processing systems," in *2017 IEEE International Conference on Big Data (Big Data)*. IEEE, 2017, pp. 1061–1068.

[58] Y. Ma and E. A. Rundensteiner, "Scalable and efficient simulation of distributed stream processing applications," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 9, pp. 2032–2046, 2018.

[59] M. Gil-Costa, L. Tapia, and J. Marin, "Asynchronous simulation protocol for stream processing platforms," *Simulation Modelling Practice and Theory*, vol. 68, pp. 123–136, 2016.

[60] S. Kamburugamuve, K. Ramasamy, M. Swany, and G. Fox, "Low latency stream processing: Apache heron with infiniband & intel omni-path," in *Proceedings of the10th International Conference on Utility and Cloud Computing*, 2017, pp. 101–110.

[61] A. S. Documentation. (2024) Apache storm documentation. [Online]. Available: https://storm.apache.org/documentation

[62] A. Toshniwal, S. Taneja, A. Shukla, K. Ramasamy, J. M. Patel, S. Kulkarni, N. Bhagat, S. Mittal, and D. Ryaboy, "Storm@twitter," in *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, 2014, pp. 147–156.

[63] IBM. (2024) Stream processing. [Online]. Available: https://www.ibm.com/cloud/learn/stream-processing

[64] J. Kreps, "Questioning the lambda architecture," *O'Reilly Radar*, 2014. [Online]. Available: https://www.oreilly.com/radar/questioning-the-lambda-architecture/

[65] N. Bonvin, T. Papaioannou, and K. Aberer, "The quest for an understandable and usable model for distributed stream processing systems," *ACM Computing Surveys (CSUR)*, vol. 52, no. 1, pp. 1–38, 2019.

[66] A. S. Foundation. (2022) Apache storm architecture. [Online]. Available: https://storm.apache.org/releases/2.3.0/Concepts.html

[67] M. Sathiamoorthy, I. Moraru, S. A. Akhlaghi, I. Gupta, and I. Stoica, "Consistency, availability, and convergence: Beyond the cap theorem," *Communications of the ACM*, vol. 58, no. 12, pp. 68–77, 2015.

[68] K. Lakshmanan, A. Das, and R. Yerneni, "Apache storm: a real-time stream processing system," *ACM SIGMOD Record*, vol. 43, no. 2, pp. 105–112, 2014.

[69] A. Inostrosa-Psijas, R. Solar, M. Marin, V. Gil-Costa, and G. Wainer, "Modeling and simulating stream processing platforms," in *2023 Winter Simulation Conference (WSC)*, 2023, pp. 3130–3141.

[70] J. Smith, "Complexity in software configuration," *Journal of Software Engineering*, 2023.

[71] J. Doe, "Usability challenges in software applications," *International Journal of Human-Computer Interaction*, 2022.

[72] P. Software, "Rabbitmq," https://www.rabbitmq.com/, 2024, accedido el 15 de febrero de 2024.

[73] S. Krug, *Don't Make Me Think: A Common Sense Approach to Web Usability*. New Riders, 2000.

[74] R. S. Pressman, *Software Engineering: A Practitioner's Approach*. McGraw-Hill Education, 2014.

[75] M. P. Papazoglou, "Service-oriented architecture: characteristics and challenges," *Information systems*, vol. 35, no. 4, pp. 357–370, 2010.

[76] S. Newman, *Building microservices*. O'Reilly Media, Inc., 2015.

[77] A. Ferscha and S. K. Tripathi, "Parallel and distributed simulation of discrete event systems," Nombre de la institución, Tech. Rep., 1998.

[78] J. Sutherland, "Scrum: The future of work," *Harvard Business Review*, vol. 92, no. 6, pp. 53–62, 2014.

[79] I. Sommerville, *Software Engineering*. Pearson, 2016.

[80] R. S. Pressman, *Software Engineering: A Practitioner's Approach*. McGraw-Hill Education, 2014.

[81] M. A. Babar, *Requirements Engineering for Software and Systems*. CRC Press, 2019.

[82] J. Smith, "User management in web applications," *Journal of Web Development*, vol. 10, no. 2, pp. 45–60, 2020.

[83] J. Doe, "Content management systems: A comprehensive guide," *Journal of Content Management*, vol. 5, no. 3, pp. 112–128, 2019.

[84] D. Brown, "Efficient event scheduling techniques for web applications," in *Proceedings of the International Conference on Web Engineering*. ACM, 2021, pp. 234–245.

[85] E. Johnson, "Advanced reporting and analytics tools for business intelligence," *Journal of Business Intelligence*, vol. 15, no. 4, pp. 78–92, 2022.

[86] M. Garcia, *Integration Patterns: Integrating External Systems with Web Applications*. Tech Publishing, 2020.

[87] M. K. Patra, A. Kumari, B. Sahoo, and A. K. Turuk, "Docker security: Threat model and best practices to secure a docker container," in *2022 IEEE 2nd International Symposium on Sustainable Energy, Signal Processing and Cyber Security (iSSSC)*, 2022, pp. 1–6.

[88] "Rabbitmq documentation: Parameters," https://www.rabbitmq.com/docs/parameters, Rabbit Technologies Ltd., accedido: 28 de Marzo 2024.

[89] N. Contributors, "Security - nestjs documentation," https://docs.nestjs.com/security, 2024, accessed: 2024-06-17.

[90] "Mongodb security checklist & best practices," https://www.mongodb.com/features/security/best-practices, MongoDB,Inc., accedido: 28 de Marzo 2024.

[91] "How to optimize docker image," https://www.geeksforgeeks.org/how-to-optimize-docker-image/, GeeksforGeeks, accedido: 28 de Marzo 2024.

[92] R. Team, "Runtime tuning - rabbitmq," https://www.rabbitmq.com/docs/runtime, 2024, accessed: 2024-10-15.

[93] "How to measure and improve node.js performance," https://raygun.com/blog/improve-node-performance/, Raygun, accedido: 28 de Marzo 2024.

[94] L. Community, "¿cuáles son las formas más eficaces de escalar una aplicación en contenedores?" https://www.linkedin.com/advice/1/what-most-effective-ways-scale-containerized-application-y4ldf, accedido: 28 de Marzo 2024.

[95] V. Acharya, "Scaling node.js applications for high traffic," https://www.linkedin.com/pulse/scaling-nodejs-applications-high-traffic-vishwas-acharya-oxdwf, accedido: 28 de Marzo 2024.

[96] Linkedin, "¿cómo se puede garantizar una alta disponibilidad en la contenedorización?" https://www.linkedin.com/advice/0/how-can-you-ensure-high-availability-containerization-gh6sf, accedido: 28 de Marzo 2024.

[97] "Disaster recovery and high availability 101," https://www.rabbitmq.com/blog/2020/07/07/disaster-recovery-and-high-availability-101, Rabbit Technologies Ltd., accedido: 28 de Marzo 2024.

[98] M. H. Ibrahim, M. Sayagh, and A. E. Hassan, "A study of how docker compose is used to compose multi-component systems," *Empirical Software Engineering*, vol. 26, pp. 1–27, 2021.

[99] R. Team, "Rabbitmq supported protocols," https://www.rabbitmq.com/protocols.html, 2024, accessed: 2024-10-15.

[100] V. Bojinov, *RESTful Web API Design with Node. js*. Packt Publishing, 2015.

[101] Docker hub. [Online]. Available: https://hub.docker.com/

[102] K. Chodorow and M. Dirolf, *MongoDB: The Definitive Guide*. O'Reilly Media, Inc., 2013.

[103] D. Merkel, "Docker: Lightweight linux containers for consistent development and deployment," *Linux Journal*, vol. 2014, 2014.

[104] J. Smith, "The benefits of docker containers," *TechToday*, 2020. [Online]. Available: https://www.techtoday.com/benefits-of-docker

[105] A. Jones, *Docker Storage Essentials*. Packt Publishing, 2019.

[106] "Visual Studio Code," https://code.visualstudio.com/, accedido: 14 febrero 2024.

[107] "Visual Studio Code Marketplace," https://marketplace.visualstudio.com/, accedido: 14 febrero 2024.

[108] Docker - build, ship, and run any app, anywhere. [Online]. Available: https://www.docker.com/

[109] T. Kubernetes, "Kubernetes," *Kubernetes. Retrieved May*, vol. 24, p. 2019, 2019.

[110] M. Chris, *Programming TypeScript*. O'Reilly Media, 2019.

[111] M. Marzolla *et al.*, "libcppsim: a simula-like, portable process-oriented simulation library in c++," in *Proc. of ESM*, vol. 4. Citeseer, 2004, pp. 222–227.

[112] A. Vakali, "Lru-based algorithms for web cache replacement," in *Electronic Commerce and Web Technologies: First International Conference, EC-Web 2000 London, UK, September 4–6, 2000 Proceedings 1*. Springer, 2000, pp. 409–418.